Verkkotapahtumien keräysjärjestelmän tietokannan uudistaminen

Petri Kotiranta

Tampereen yliopisto
Informaatiotieteiden yksikkö
Tietojenkäsittelyoppi
Pro gradu -tutkielma
Ohjaajat: Timo Poranen ja Marko Junkkari
Toukokuu 2015

Tampereen yliopisto
Informaatiotieteiden yksikkö
Tietojenkäsittelyoppi
Tekijän nimi: Petri Kotiranta
Pro gradu -tutkielma, 66 sivua
Toukokuu 2015

Tietokannoissa on datamäärien kasvaessa alettu suosia hajautettuja tietokantaratkaisuja. Nykyään käsiteltävän dataliikenteen määrä on niin suuri, että yksi tietokantapalvelin ei pysty vastaamaan suuren liikennemäärän aiheuttamaan kuormaan. Viime vuosina onkin nähty monien NoSQL- ja NewSQL-tietokantaratkaisuiden esiinmarssi. Nämä tietokannat tarjoavat perinteisistä SQL-tietokannoista poikkeavia tietomalleja ja mahdollistavat tietokannan hajauttamisen useamman palvelimen klusterikokoonpanolle. Tietokantoja on markkinoilla useita ja ne on suunniteltu eri tarkoituksiin.

Tässä tutkimuksessa luodaan katsaus NoSQL- ja NewSQL-tietokantoihin ja esitellään erään yrityksen verkkotapahtumien keräysjärjestelmä. Yrityksessä on kiinnostuttu uusista tietokantaratkaisuista, sillä nykyinen SQL-tietokanta ei skaalaudu hyvin suurille datamassoille. Tutkimuksessa tarkastellaan miten NoSQL- ja NewSQL-tietokannat soveltuisivat verkkotapahtumien keräysjärjestelmän tietokannaksi. Lopussa valitaan testattavaksi kaksi tietokantaa, joiden suorituskykyä ja ominaisuuksia tutkitaan tarkemmin.

Avainsanat ja -sanonnat: NoSQL, NewSQL, ACID, BASE, CAP, Avain-arvo-varasto, Dokumenttivarasto, Sarakeperhevarasto, Graafitietokanta.

Sisällysluettelo

Termit ja niiden määritelmät

1:N-suhde	Yhden tietueen suhde moneen.
2G	Toisen sukupolven matkapuhelinstandardit, joihin kuuluvat GSM ja GPRS.
3G	Kolmannen sukupolven matkapuhelinstandardi, johon kuuluvat HSPA, W-CDMA ja UMTS.
4G	Neljännen sukupolven matkapuhelinteknologia.
BSC	Base Transceiver Station on mobiiliverkossa toimiva radiovastaanotin, joka vastaa yhteydestä matkapuhlimien ja tukiaseman välillä.
CAPEX	Hankkimiseen liittyvät kustannukset.
CFX-5000	Sessionhallintaentitetti IP-pohjaisissa verkoissa.
Coda-tiedostojärjestelmä	Hajautettu tiedostojärjestelmä Linuxille, NetBSD:lle ja FreeBSD:lle.
DX-HLR	Home Location Register on tietokanta, joka sisältää listan niistä tilaajista, joilla on oikeus käyttää GSM-verkkoa.
EnodeB	Evolved Node B on kehittyneempi versio UMTS-verkoissa käytetystä Node B –elementistä.
FlexiNG	Flexi Network Gateway on mobiiliverkoille suunniteltu yhdyskäytävä.
FlexiNS	Flexi Network Server on palvelinratkaisu mobiilidatan hallintaan.
GPRS	General Packet Radio Service on pakettikytkentäinen tiedonsiirtopalvelu, joka toimii GSM-verkoissa.
GSM	Global System for Mobile Communications, yleisesti käytössä oleva matkapuhelinjärjestelmä.
HSPA	High Speed Packet Access on matkapuhelinviestinnässä käytettävien protokollien kokoelma, joka parantaa kolmannen sukupolven verkon suorituskykyä.
Klusteri	Useamman eri palvelintietokoneen muodostama kokonaisuus.
LDAP	Lightweight Directory Access Protocol on protokolla, jota käytetään hakemistopalvelujen käytössä. Useimmiten LDAP:a käytetään käyttöoikeuksien tarkistamiseen.
LTE	Long Term Evolution on edistynyt tekniikka, jolla pyritään kasvattamaan verkkoliikenteen nopeutta.
M:N-suhde	Useamman tietueen suhde useampaan tietueeseen.
MGW	Media gateway on käännöslaite, joka konvertoi mediavirtoja eri verkkojen välillä.
Monikko	Relaatiotietokantojen tietue (taulun rivi).
MSC/MSS	Mobile Switcing Centre on puhelinten palvelupyyntöjä välittävä keskus, joka reitittää puheluita tukiasemien välillä.
OpenTAS	Open Telecom Application Server on palvelin, joka tarjoaa täydentäviä palveluita ääni, viesti ja videosessioihin.
OPEX	Ylläpitoon liittyvät kustannukset.
RAB	Radio Access Bearer on radioyhteyden välittäjä, joka välittää dataa käyttäjän pyytämille palveluille.
RNC	Radio Network Controller on elementti, joka hallinnoi NodeB-elementtejä UMTS-verkossa.

RRC	Radio Resource Control hallinnoi signalointia UMTS-verkoissa.
SGSN	Serving GPRS support node on vastuussa pakettipohjaisesta dataliikenteestä GPRS-verkossa.
Solmu	Yksi palvelintietokone, joka kuuluu klusteriin.
Suurtietokone	Yleensä suurikokoinen palvelintietokone, jossa on paljon suorituskykyä ja tätä käytetään suurien datamäärien prosessointiin.
Säie	Ajossa olevan tietokoneohjelman osa, joka käyttää ohjelman resursseja. Säikeitä voidaan ajaa ohjelmassa useita kappaleita rinnakkain.
Transaktio	Tietokannoissa transaktiolla tarkoitetaan tietokannassa tapahtuvia muutoksia. Esimerkiksi muutoksia tietokannassa olevaan dataan.
UMTS	Universal Mobile Telecommunications System on kolmannen sukupolven matkapuhelinteknologia.
W-CDMA	Wideband Code Division Multiple Access on rajapinta, joka määrittelee miten mobiililaitteiden kommunikoinnin tukiasemian kanssa ja miten signaalit moduloidaan.
WiMAX	Worldwide Interoperability for Microwave Access on standardi, jonka avulla pyritään saavuttamaan laajakaistaverkon tasoisia nopeuksia matkapuhelinverkoissa.

1 Johdanto

Tietokantojen historiassa tietokanta on useimmiten tarkoittanut relaatiotietokantaa. Relaatiotietokannat ja niissä käytettävä SQL-kyselykieli ovat olleet pitkään käytössä monien eri organisaatioiden tietojärjestelmissä. Relaatiotietokantojen käyttämä tietomalli on osoittautunut erittäin luotettavaksi ja monipuoliseksi useammanlaiseen käyttöön, joten se on pitänyt asemansa viimeiset vuosikymmenet.

Nykyään Internetin ja älypuhelinten myötä on tultu siihen tilanteeseen, että tietokantaan tallennettavan datan määrä on moninkertaistunut. Aikaisemmin tietokannat asennettiin suurtietokoneille ja niihin vietiin tietoa vain organisaation sisällä. Nykyään tietokantoja käytetään yhä useammin verkkosivustojen tietosisällön säilyttämiseen ja monelle verkkosivustolle, esimerkiksi Wikipediaan tai Facebookiin voi kuka tahansa lisätä tietoa. Näin verkkosivustojen pohjalla olevat tietokannat joutuvat käsittelemään suuria datamääriä ja tietokannan pitää pystyä käsittelemään dataa tarpeeksi nopeasti, jotta verkkosivusto pysyy käytettävänä.

Perinteinen relaatiotietokantamalli on suunniteltu aikana, jolloin Internetiä ja matkapuhelimia ei ollut vielä yleisessä käytössä. Relaatiotietokantojen perinteisiä käyttäjiä ovat olleet esimerkiksi pankit ja pankkien tarkoituksiin relaatiotietokannan eheä tietomalli onkin sopinut todella hyvin. Tosin nyt Internetin ja älypuhelinten myötä on tullut ongelmaksi relaatiotietokantojen skaalatutuvuus. Relaatiotietokannat on suunniteltu asennettavaksi yhdelle tietokoneelle. Tämä takaa tietokannan eheyden. Internet-sivustoille tulee kuitenkin usein niin paljon dataa, että tehokaskaan tietokone ei pysty yksin käsittelemään tätä kaikkea datamassaa. Tarvittaisiinkin useamman tietokoneen klustereita, jotta datamassan aiheuttama kuorma saataisiin tehokkaasti käsiteltyä. Perinteiset relaatiotietokannat skaalautuvat useamman tietokoneen hajautetulle klusterille huonosti ja näin onkin nähty uusien NoSQL- ja NewSQL-tietokantojen esiinmarssi. Näissä tietokannoissa keskeisimpinä uusina ominaisuuksina ovat uudet, perinteisistä tietokannoista poikkeavat tietomallit ja skaalautuvuus useamman palvelimen muodostamalle klusterille.

Tämä tutkimus on katsaus uusiin NoSQL- ja NewSQL-tietokantaratkaisuihin ja erään yrityksen verkkotapahtumien keräysjärjestelmään, jossa käytetään tällä hetkellä perinteistä relaatiotietokantaa. Yrityksessä haluttaisiin etsiä korvaajaa nykyiseen relaatiotietokantaan pohjautuvalle ratkaisulle NoSQL- ja NewSQL-tietokannoista.

Tutkimuskysymyksiä ovat:

1. Miten NoSQL- ja NewSQL-tietokannat eroavat perinteisistä tietokannoista?
2. Miten NoSQL- ja NewSQL-tietokannat täyttävät verkkotapahtumien keräysjärjestelmän asettamat vaatimukset?
3. Tarjoavatko testiin valitut NoSQL- ja NewSQL-tietokantaratkaisut parempaa suorituskykyä nykyiseen tietokantaratkaisuun nähden?

Kysymyksiin 1 ja 2 on tässä tutkimuksessa lähdetty hakemaan vastauksia kirjallisuuskatsauksen avulla. NoSQL- ja NewSQL-tietokannoista ja niiden ominaisuuksista on pyritty etsimään tietoa tutkimusartikkelien ja tietokantojen omien Internet-sivujen kautta. Kysymystä 3 varten valittiin testiin kaksi verkkotapahtumien keräysjärjestelmän kannalta mielenkiintoista tietokantatoteutusta. Niitä varten toteutettiin testiajuri, jolla tietokantaan ajettiin verkkotapahtumien keräysjärjestelmässä käytettävää dataa ja tutkittiin kuinka nopeasti uudet tietokantaratkaisut pystyvät vastaanottamaan ja kyselemään tätä dataa.

Luvussa 2 käydään läpi perinteisiä tietokantamalleja. Luvussa käydään lyhyesti läpi relaatiotietokantoja edeltäneet hierarkkinen tietokantamalli ja verkkotietokanta ja sen jälkeen varsinainen relaatiotietokanta ja lyhyesti oliotietokanta. Relaatiotietokannoista kuvataan niissä keskeiset ACID-säännöt, SQL-kieli ja yleisimmät rajapinnat. Lisäksi käydään läpi muutama yleisimmin käytössä oleva relaatiotietokanta.

Luvussa 3 käsitellään uusia NoSQL- ja NewSQL-tietokantoja. Niistä käydään läpi näille tietokannoille keskeinen BASE-malli ja hajautettujen tietokantojen piirteitä kuvaava CAP-teoreema. NoSQL-tietokannoista käydään läpi niiden erilaiset tietomallit ja rajapinnat.

Luvussa 4 esitellään verkkotapahtumien keräysjärjestelmä. Tässä käydään läpi keräysjärjestelmän arkkitehtuuri ja järjestelmän vaatimukset sen käyttämälle tietokannalle. Lisäksi kerrotaan nykyisistä ratkaisuista näiden vaatimusten toteuttamiseksi.

Luvussa 5 esitellään kaksi tietokantaa, jotka valittiin testattavaksi. Luvussa käydään läpi ensin aikaisempia tutkimuksia NoSQL- ja NewSQL-tietokannoista, esitellään valitut tietokannat ja käydään läpi kuinka näiden tietokantojen avulla voisi toteuttaa verkkotapahtumien keräysjärjestelmän arkkitehtuurin mukaisen tietokantaratkaisu.

Luvussa 6 esitellään valittuihin tietokantoihin tehdyt tutkimukset ja niitä varten toteutettu sovellus InserterDriver. Tutkimuksista esitellään datan syöttämistä tutkivat testit ja testikyselyt vietyyn datamassaan kummallakin tietokannalla. Lopuksi esitetään yhteenveto ja johtopäätökset luvussa 7.

2 Perinteiset tietokannat

2.1 Hierarkkinen tietokanta ja verkkotietokanta

Ennen relaatiotietokantojen voittokulkua käytettiin hierarkkista tietokantamallia ja verkkotietokantamallia. Molemmissa näistä malleista tietokanta koostuu tietueista, joiden välillä on linkkejä. Tietue sisältää varsinaisen datan ja se voi sisältää useampia arvoja. Tietueiden väleillä on suorat linkit toisiin tietueisiin. Näin tietokannasta tulee rakenteeltaan tietueiden välinen verkosto.

Hierarkkinen tietokantamalli kehitettiin 1960-luvulla ja se oli ensimmäisiä tietokantamalleja tietokantojen historiassa. Aikansa suurtietokoneet kehitettiin siten, että niissä otettiin käyttöön tämä malli. Hierarkkisessa tietokantamallissa tietueet tallennetaan puumaiseen rakenteeseen. Jokaisella tietueella tässä rakenteessa voi olla yksi vanhempi ja monta lasta. Tämä tietomalli on hyvä 1:N-tyyppisiä suhteita sisältävälle datalle, mutta N:M-tyyppisissä relaatioissa joudutaan käyttämään virtuaalitietueita tai tallentamaan dataa useaan kertaan. [Elmasri and Navathe, 1989 ss. 253–276]

Verkkotietokannan malli julkaistiin ensimmäisen kerran Database Task Groupin raportissa vuonna 1971 [CODASYL, 1971]. Toisin kuin hierarkkisessa tietokantamallissa, verkkotietokantamallissa sallitaan, että tietueella on useampi vanhempi. Tämän myötä verkkotietokanta on parempi M:N-tyyppisiä suhteita sisältämälle datalle ja siinä ei tarvitse tallentaa dataa useaan kertaan. 70-luvulla julkaistiin myös relaatiotietokantamalli, joka lopulta syrjäytti hierarkkisen tietokantamallin ja verkkotietokantamallin.

Kummassakin näistä vanhoista tietokantamalleista tietoa haetaan navigoimalla tietueiden välille määriteltyjen suhteiden kautta. Yhtenäistä kyselykieltä ei ole, vaan kyselyt pitää toteuttaa jollakin ohjelmointikielellä. 60- ja 70-luvuilla oli suosittua toteuttaa kyselyt COBOL-kielellä. Tietueisiin ja linkkeihin pohjautuvista tietokantamalleista saattaa tulla paljon monimutkaisempia kuin relaatiotietokannoissa. Kun dataa on paljon, tietueita ja niiden välisiä linkkejä on suuri määrä. Jos tietoa haluaa poistaa tai lisätä kantaan on tietueen lisäksi poistettava tai lisättävä myös kaikki linkit toisiin tietueisiin. Tietokannan käsittely saattaa olla muutenkin hankalaa, sillä tietoa haetaan navigoimalla linkistä toiseen. Toisaalta näissä kannoissa saattaa olla mahdollista tehdä nopeita kyselyitä, sillä hakeminen tehdään jonkin ohjelmointikielen avulla ja hakemisprosessi on täysin ohjelmoijan hallittavissa. Täten ohjelmoijalla on mahdollisuus optimoida haut varsin hyvin.

2.2 Relaatiotietokanta

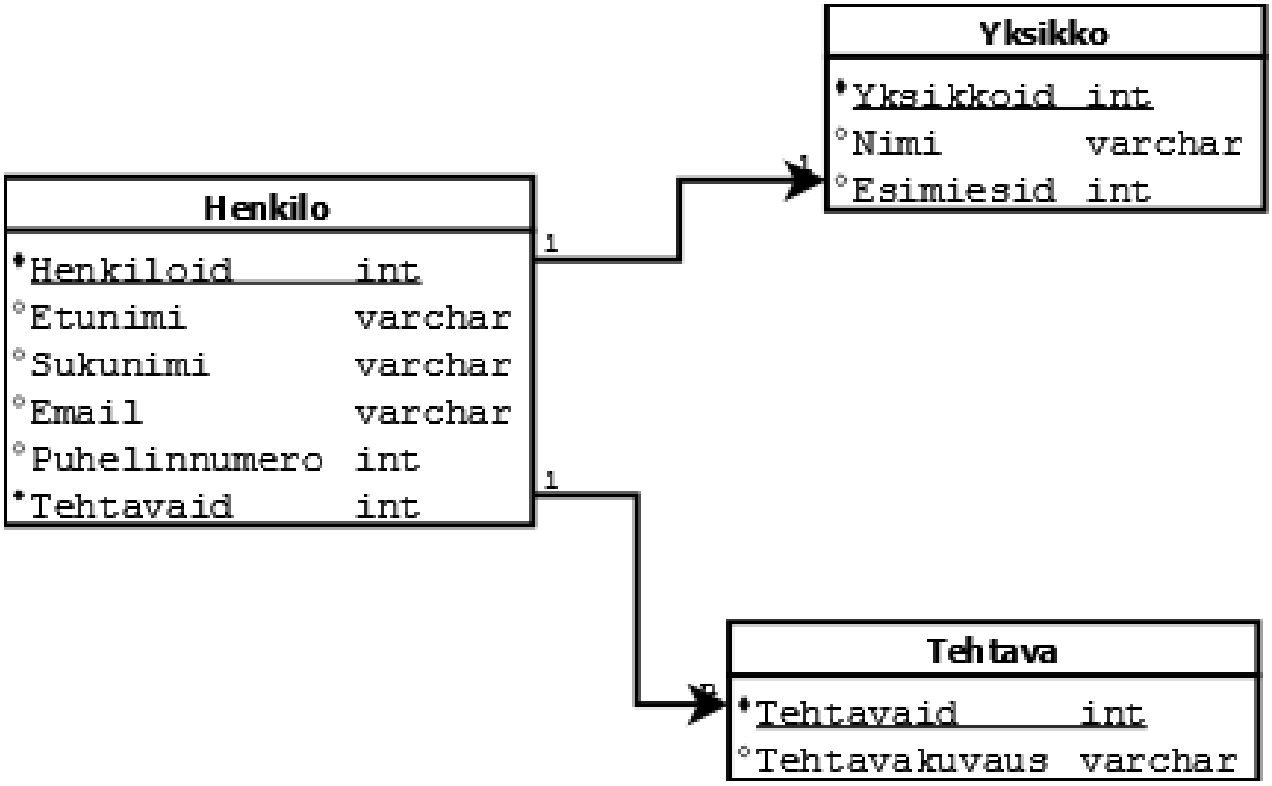
Relaatiomallin historia alkaa, kun IBM:llä työskennellyt matemaatikko Edgar F. Codd julkaisi sitä käsittelevän kirjoituksen vuonna 1970 [Codd, 1970]. Relaatiomalli oli vallankumouksellinen, sillä toisin kun hierarkkisessa mallissa ja verkkotietokantamallissa, tietomalli koostui tietueiden ja niiden välisten linkkien sijasta relaatioista(tauluista) ja niiden välisistä suhteista [Fortune, 2014]. Relaatiomallissa data erotettiin sitä käsittelevästä ohjelmasta ja datan käsittelyssä alettiin käyttää omaa kyselykieltä.

Ensimmäinen relaatiotietokantaa hyödyntävä järjestelmä oli IBM:n System R, jota kehitettiin 1974–1979 välillä [Astrahan et al., 1976]. Samalla kehitettiin myös Structured Query Language eli SQL-kyselykieli, jonka avulla voidaan käsitellä tietokannassa olevaa dataa [Chamberlin et al., 1974]. 1980-luvulla alkoi relaatiotietokantojen voittokulku ja nykyään relaatiotietokannat ovat syrjäyttäneet muut tietokantatyypit. Tietokannalla tarkoitetaankin nykyään yleensä juuri relaatiotietokantaa, josta haetaan tietoa SQL-kyselykielen avulla.

Relaatiotietokantojen vahvuutena on standardoitu SQL-kieli, jota kaikki markkinoilla olevat relaatiotietokannat käyttävät. Tietomalli on myös selkeä, helposti ymmärrettävä ja eheä. Toisaalta relaatiotietokannat skaalatuvat huonosti suurille datamäärille. Erityisesti, jos hajautetaan dataa useille eri palvelimille. Relaatiotietokannoissa on ollut niiden käyttöhistorian ajan käytössä ”one size fits all”-ajattelu, eli samaa tietokantatyyppiä käytetään useantyyppisille erilaisille sovelluksille [Stonebraker and Çetintemel, 2005].

2.3 Relaatiotietokannan rakenne

Relaatiotietokannassa tieto säilötään kaksiulotteisiin tauluihin, jonka rivit muodostavat tietueita. Tietue koostuu kentistä, joihin tallennetaan varsinainen tieto. Tallennettavalla tiedolla voi olla eri tietotyyppejä. Se voi olla numeromuotoista (integer, float…), merkkitietoa (char, varchar…), päiväystietoa, (data, time, datetime), totuusarvoja tai se voi ilmaista loogista tilaa. Taulukoiden kentät muodostavat monikoita, joissa voi olla vierekkäin useanlaisia tietotyyppejä. [Elmasri and Navathe, 2010, ss. 57–85]



Kuva 1: Yksinkertainen relaatiotietokannan kaavio.

Kuvassa 1 nähdään yksinkertaistettu esimerkki relaatiotietokannan kaaviosta. Tässä esimerkissä tietokannassa on Henkilo-taulu, Yksikko-taulu ja Tehtava-taulu. Henkilo-tauluun listatuilla henkilöillä on jokaisella oma yksikäsitteinen tunniste, joka on tässä tapauksessa Henkiloid. Tämä attribuutti toimii taulun pääavaimena ja sen on oltava jokaisella henkilöllä yksikäsitteinen. Henkiloid tunnistaa jokaisen henkilön ja tätä käytetään viitatessa Yksikko-tauluun. Yksikko-taulussa on listattu työyksiköitä ja jokaisella työyksiköllä on oma esimies. Yksiköiden esimiesten tiedot haetaan Henkilo-taulusta siten, että esimiesid:tä käytetään viittaamaan Henkilo-taulun henkiloid:hen. Henkiloillä voi olla tehtäviä ja tehtävien tiedot löytyvät tehtävä-taulusta. Tehtävät tunnistetaan tehtavaid:n avulla ja vastaava id löytyy myös Henkilo-taulusta. Kun Henkilo-taulu ja Tehtava-taulu yhdistetään, Tehtavaid:n perusteella saadaan haettua jokaisen henkilön kaikki tehtävät.

2.4 SQL-kyselykieli

Kuten kohdassa 2.2 mainittiin, SQL-kyselykieli kehitettiin ensimmäisen relaatiotietokantajärjestelmän, System R:n, yhteydessä. SQL:n ensimmäistä versiota kutsuttiin nimellä SEQUEL eli Structured English Query Language [Chamberlin et al., 1974]. Nimi lyhennettiin kuitenkin SQL-muotoon, sillä SEQUEL oli jo rekisteröity tavaramerkki eräälle lentokonevalmistajalle. SQL-kieli tarjoaa menetelmät

tiedon käsittelemiselle relaatiotietokannassa. SQL-kieli voidaan jakaa tiedon määrittelykieleen (data definiton language) ja tiedon manipulointikieleen (data manipulation language) [Rockoff, 2011]. Tietokannan luontiin liittyvät komennot SQL:ssä liittyvät datan määrittelyyn ja tiedon hakuun ja käsittelyyn liittyvät lauseet taas datan manipulointiin.

SQL-kielen avulla relaatiotietokannassa olevaa tietoa voidaan hakea, sitä voidaan yhdistellä, muokata, poistaa ja lisätä. Tärkeimmät käskyt SQL-kielessä ovat SELECT, UPDATE, INSERT, DELETE ja CREATE. SELECT-käskyn avulla voidaan hakea jotakin tietoa tietyistä tauluista, UPDATEN avulla voidaan päivittää alkioiden arvoja, INSERT-käskyllä voidaan lisätä alkioiden arvoja. DELETE-käskyllä poistetaan arvoja ja CREATE-käskyllä voidaan luoda uusia tauluja. Näiden lisäksi tauluja voidaan yhdistellä JOIN-lauseita apuna käyttäen ja taulujen arvoja voidaan ryhmitellä GROUP BY -lauseilla. Ryhmittelyssä voidaan tehdä laskuoperaatioita sarakkeen arvoihin aggregointifunktioiden avulla. [Elmasri and Navathe, 2011]

SQL on varsin hyvin standardoitu kieli. Ensimmäinen revisio SQL:stä standardoitiin vuonna 1986 nimellä ”SQL-86”. Vuonna 1992 alkuperäiseen standardointiin lisättiin paljon uusia nykyään tavallisessa käytössä olevia ominaisuuksia [ISO/IEC 9075, 1992]. Tietotyypeistä uusia olivat aikaan liittyvät tietotyypit ja merkkijonot. Uusina operaatioina olivat matemaattiset lausekkeet, ehtolausekkeet ja liitoslauseet. Kieleen lisättiin uusia käskyjä liittolauseille, tapauslauseille (engl. case) ja tyyppimuunnoksille (engl. cast). Vuoden 1999 standardi lisäsi tuen säännöllisille lausekkeille (engl. regular expression), rekursiivisille kyselyille, liipaisimille, proseduraalisille ja kontrollivuolauseille, ei-skalaareille tyypeille ja joillekin olio-ominaisuuksille [ISO/IEC 9075(4,6), 1999].

Vuonna 2003 standardiin lisättiin XML-merkintäkieleen liittyviä ominaisuuksia, ikkunafunktiot, standardoidut sekvenssit ja sarakkeet automaattisesti generoiduilla arvoilla [ISO/IEC 9075(4,14), 2003]. Vuoden 2006 standardi paransi yhteensopivuutta XML-merkintäkielen kanssa. Tässä standardiversiossa määritellään tavat tuoda ja tallentaa XML-pohjaista dataa tietokantaan, XML-pohjaisen datan käsittely tietokannassa ja XML-pohjaisen ja SQL-pohjaisen datan julkaisu XML-muodossa [ISO/IEC 9075(14), 2006]. Vuoden 2008 standardi sallii ORDER BY -lauseen kursorimääritteiden ulkopuolella, lisää INSTEAD OF -liipaisimet ja lisää myös TRUNCATE-lauseen osaksi standardia [ISO/IEC 9075(4), 2008]. ORDER BY -lausetta käytetään järjestelemään haun tuloksena saatu joukko jonkin sarakkeen mukaan. INSTEAD OF -liipaisinten avulla voidaan käyttää tietokannan näkymiä tauluina ja niihin voidaan ajaa INSERT-, UPDATE- ja DELETE-operaatioita.

TRUNCATE-lauseen avulla voidaan tyhjentää koko taulun sisältö DELETE-lauseen tavoin, tosin monissa tietokannoissa TRUNCATE on DELETE-lausetta tehokkaampi.

Viimeisimmässä, vuoden 2011 standardissa pääasiallisimpana uudistuksena on tuki ajallista dataa sisältäville tietokannoille. Kieleen lisättiin muun muassa joukko predikaatteja, kuten OVERLAPS ja PRECEDES, ajallisten arvojen käsittelemiseksi [ISO/IEC 9075(4), 2011]. OVERLAPS-predikaatin avulla voidaan ratkaista sellaisia aikaleimoihin liittyviä ongelmia, joissa aikaleimat ovat keskenään päällekkäisiä. PRECEDES-predikaatilla voidaan tutkia, jos jokin aikaleima edeltää toista aikaleimaa.

2.5 ACID

ACID on teoreema, jonka mukaan määritellään tietokannan transaktioiden eheys. Akronyymi ACID tulee sanoista atomisuus (engl. atomicity), eheys (engl. consistency), eristyneisyys (engl. isolation) ja pysyvyys (engl. durability) [Haerder and Reuter, 1983]. Atomisuus tarkoittaa, että transaktio joko suoritetaan kokonaan tai sitä ei suoriteta ollenkaan. Eheys tarkoittaa, että tietokannan tila pysyy eheänä transaktion suorituksen jälkeen. Eristyvyydellä tarkoitetaan, että transaktiota suoritettaessa muut tapahtumat eivät saa vaikuttaa siihen. Pysyvyys tarkoittaa, että transaktion tekemät muutokset tietokantaan ovat pysyviä. Nykyään kaikki relaatiotietokannat pyrkivät noudattamaan ACID-vaatimuksia. Jos kyseessä on tietokantaklusteri ja tietokannan transaktiot ovat hajautettuja, käytetään yleensä kaksivaiheista vahvistusprotokollaa (engl. two phase commit protocol) ACID-ominaisuuksien toteuttamiseksi. Kaksivaiheinen vahvistusprotokolla voidaan jakaa äänestysvaiheeseen ja päätösvaiheeseen. Yhtä klusterin solmuista kutsutaan koordinaattoriksi, joka aloittaa transaktion klusterin sisällä.

Jos häiriöitä ei tapahdu, kaksivaiheinen vahvistaminen (engl. two phase commit) etenee seuraavasti: (1) koordinaattorina toimiva solmu lähettää äänestyspyynnön kaikille muille solmuille, (2) kun solmu saa äänestyspyynnön, se vastaa siihen joko myöntävästi tai kieltävästi, (3) koordinaattori kerää äänet kaikilta solmuilta ja jos kaikki vastaavat kyllä, koordinaattori päättää vahvistaa ja lähettää viestit vahvistuksesta kaikille osallistujille. Muissa tapauksissa koordinaattori päättää peruuttaa tapahtuman ja lähettää peruutusviestit kaikille osallistujille, jotka äänestivät kyllä. (4) Kaikki solmut, jotka äänestivät kyllä, odottavat vahvistus- tai peruutusviestiä koordinaattorilta. Solmu vahvistaa tai peruuttaa ja tämän jälkeen kaksivaiheinen vahvistaminen päättyy. Kaksivaiheisen vahvistamisen tarkoituksena on siis taata, että hajautetun tietokannan solmut pysyvät keskenään synkronoituna. [Bernstein et al., 1987]

2.6 Relaatiotietokantojen rajapinnat

Jos tietokantaa halutaan käyttää tietokannan ulkopuolisilla sovelluksilla, se tehdään yleensä rajapinnan kautta. Rajapinta toimii kommunikointikanavana tietokannan ja sitä käyttävän sovelluksen välillä. Yleensä tietokantaa käyttävä ohjelma kirjoitetaan siten, että siinä toteutetaan jonkin rajapinnan määrittämät funktiot, joiden avulla SQL-lauseita saadaan ajettua tietokantaan. SQL-kielelle on asemansa vakiinnuttanut käytännössä kaksi rajapintaa ODBC ja JDBC.

2.6.1 ODBC

Lyhenne ODBC tulee sanoista Open Database Connectivity. Se on Microsoftin kehittämä standardoitu avoin rajapinta tietokannoille. ODBC muodostaa siis linkin tietokannan ja sovellusten välille siten, että sovelluksen ja tietokannan tarvitsee tuntea vain ODBC-yhteyteen tarvittavat komennot. Rajapinta on täysin riippumaton ohjelmointikielestä, käytettävästä tietokannasta ja käyttöjärjestelmästä. Näin se mahdollistaa pääsyn tietokantaan miltä tahansa sovellukselta riippumatta siitä, mikä tietokantajärjestelmä on käytössä. ODBC:n ensimmäinen versio 1.0 julkaistiin vuonna 1992 ja se oli ensimmäisiä rajapintoja SQL-tietokannoille. ODBC on nykyään varsin yleisesti käytössä ja sitä tukee suurin osa nykyisistä tietokannoista ja tietokantaa käyttävistä sovelluksista. [Geiger, 1995]

2.6.2 JDBC

JDBC tulee sanoista Java Database Connectivity. Se on ODBC:n kaltainen ohjelmointirajapinta Java-ohjelmointikielelle. Javan kehittäjä Sun Microsystems julkaisi vuonna 1996 JDBC 1.0:n osana Javan kehitystyökaluja, jonka jälkeen se on ollut osana Javaa. JDBC kehitettiin ODBC:n pohjalta varta vasten Java-ohjelmointikielelle. Vaikkakin ODBC on periaatteessa riippumaton ohjelmointikielestä, se on suunniteltu etupäässä alustariippuvaisille ohjelmointikielille. Java on taas alustariippumaton, joten ODBC:n käytössä saattaa esiintyä ongelmia Javan kanssa. ODBC:n arkkitehtuuri on myös varsin monimutkainen ja se saattaa olla ohjelmoijalle hankala JDBC:n verrattuna. Yhteensopivuus ODBC:n kanssa toteutetaan käyttämällä JDBC–ODBC-siltoja. [Reese, 2000]

2.7 Yleisimmät relaatiotietokannat

Itävaltalaisen Solid IT -konsultointiyrityksen ylläpitämän DB-Engines-listan relaatiotietokantojen kärkiviisikosta löytyy tätä tutkimusta tehdessä ensimmäisenä Oracle [2015], toisena MySQL [2015], kolmantena Microsoft SQL Server [2015], neljäntenä PostgreSQL [2015] ja viidentenä DB2 [2015] [DB-Engines, 2015a]. Nämä kannat ovat yleisimmät käytössä olevat tietokannat tätä työtä kirjoitettaessa. Näistä kannoista Oracle, Microsoft SQL Server ja DB2 ovat suljetun lähdekoodin tietokantoja ja MySQL ja PostgreSQL ovat avoimen lähdekoodin tietokantoja.

Oracle ja DB2 ovat viisikosta vanhimpia. Kummankin historia palautuu aikaisemmin kohdassa 2.2 mainittuun System R -toteutukseen. Oraclen tietokanta on yksi ensimmäisistä kaupallisesti saataville tuoduista relaatiotietokannoista. Tietokannan kehitys aloitettiin vuonna 1977 Codd:n tutkimuspaperin ja System R:n innoittamana. Vaikkakin IBM kehitti ensimmäisenä relaatiotietokantatoteutuksena pidettyä System R -järjestelmää, Oracle ehti kuitenkin markkinoille ensin. IBM:llä nimittäin keskityttiin 70-luvulla relaatiotietokantoja enemmän verkkotietokantojen ja hierarkkisten tietokantojen kehitykseen. IBM:n vastine Oraclelle eli DB2 julkaistiin vuonna 1983. Kolmas suosittu suljetun lähdekoodin tietokanta viisikosta, Microsoft SQL Server, tuli markkinoille vasta vajaa kymmenen vuotta myöhemmin.

Oracle ja DB2 toimivat yleisimmillä käyttöjärjestelmillä, kun taas Microsoftin SQL Server toimii vain Microsoftin käyttöjärjestelmillä. Oraclea ja DB2:ta käytetään yleensä isoissa ja tehtäväkriittisissä (engl. mission-critical) järjestelmissä. DB2 on useimmiten käytössä IBM:n omissa suurtietokoneissa [W3Computing, 2015]. SQL Serveriä taas käytetään Microsoftin pienissä ja keskisuurissa palvelimissa. Viisikosta PostgreSQL ja MySQL tulivat markkinoille 1990-luvun puolivälissä. Siinä missä MySQL on tullut tunnetuksi nopeana tietokantana, PostgreSQL on tullut tunnetuksi ominaisuuksiltaan parempana ja luotettavampana tietokantana [Ehringer, 2014]. Koska molemmat näistä tietokannoista ovat ilmaisia, ne tarjoavat helpon tavan tietokantojen käyttöönotolle. Kääntöpuolena kaupallisiin suljetun lähdekoodin tietokantoihin nähden on tukipalveluiden puute.

2.8 Oliotietokannat

Relaatiotietokantojen korvaajaksi arveltiin aikanaan oliotietokantoja [Ullman, 1988; Baker, 1992]. Oliotietokannoissa lähestymistapa datan organisointiin muistuttaa olio-ohjelmointia. Tämä lähestymistapa sopii paremmin yhteen nykyisissä ohjelmointikielissä käytetyn olioparadigman kanssa. Oliotietokannassa tietomalli nähdään joukkona olioita, joissa tietosisältö on tallennettu attribuutteihin ja attribuuteilla on niitä käsittelevät metodit. Vaikkakin malli sopii paremmin yhteen yleisesti käytössä olevan olio-ohjelmointimallin kanssa, oliotietokannat eivät ole kuitenkaan syrjäyttäneet relaatiotietokantoja. Oliotietokantojen ongelmana on ainakin vakiintuneen käsittelymallin puute. Standardoitua ODMG-mallia on kyllä yritetty kehittää ja sen viimeisin versio on 3.0 [Cattell and Barry, 2000]. Malli ei kuitenkaan yleistynyt. Relaatiotietokantamalli on osoittautunut riittäväksi moniin sovelluksiin ja oliopohjaisen mallin etuja ei ole koettu tarpeelliseksi. Oliomallilla on kuitenkin käyttöä tietyillä sovellusalueilla, esimerkiksi CAD-mallinnuksessa.

3 NoSQL- ja NewSQL-tietokannat

3.1 Yleistä

NoSQL-tietokannat ovat uusi perhe erilaisia tietokantasovelluksia, joiden tietomalli poikkeaa perinteisten tietokantojen relaatiomallista. NoSQL-liikehdinnän taustalla on käsiteltävän datan määrän raju lisääntyminen, kuten johdantoluvussa mainittiin. Esimeriksi vuonna 2008 Google prosessoi enemmän kuin 20 petatavua dataa päivässä [Dean and Ghemawat, 2008]. Suuret datamäärät vaativat entistä tehokkaampia ratkaisuja, jotta kaikki saadaan käsiteltyä. Perinteisten SQL-tietokantojen heikkoutena on relaatiomallin jäykkyys, sillä se on suunniteltu rakenteiselle datalle. Esimerkiksi pankin tietojärjestelmät ja organisaatioiden henkilörekisterit ovat olleet SQL-tietokantojen vakiintuneita sovellusalueita. Nykyaikaiset kansainväliset verkkopalvelut siirtävät kuitenkin huomattavasti enemmän dataa ja verkkopalvelun sisältämällä datalla ei välttämättä ole yhtenäistä rakennetta, joten kaikkia perinteisten SQL-tietokantojen ominaisuuksia ei tarvita ja ne voivat toimia hidasteena. NoSQL-tietokantojen yhteisenä piirteenä onkin SQL:stä tutun tietomallin jättäminen ja sen korvaaminen jollain tarkoitukseen paremmin sopivalla yksinkertaistetulla mallilla.

NoSQL-termiä käytti ensimmäistä kertaa Carlo Strozzi vuonna 1998 kehittämästään tietokannasta, jonka rajapinta poikkesi standardista SQL-rajapinnasta [Strozzi, 2010]. Useimmiten NoSQL tulkitaan tulevan sanoista ”Not only SQL” eli ei pelkästään SQL. Toisen kerran termiä käytti Eric Evans vuoden 2009 konferenssissa, jossa haluttiin löytää yhteinen nimi uusille tietokannoille, joiden ominaisuudet poikkesivat perinteisestä SQL-mallista [Evans, 2009]. Tästä lähtien termiä NoSQL on käytetty yhteisenä nimikkeenä useille erilaisille 2000-luvulla aloitetuille SQL-tietokannoista eroaville tietokantaprojekteille. Nykyään löytyykin jo noin 150 tietokantaa, jotka luokitellaan NoSQL-termin alle [Edlich, 2009].

NoSQL-tietokannoissa ei ole ”one size fits all” -ajattelua, kuten relaatiotietokannoissa, vaan tietokantojen ominaisuudet ovat räätälöityjä johonkin tiettyyn tarkoitukseen. Lisäksi niissä on useimmiten luovuttu ACID-ominaisuuksista, sillä ACID-ominaisuuksien vaatimukset ovat liian tiukkoja, mikäli käytetään useamman tietokantapalvelimen muodostamaa klusteroitua palvelinratkaisua. Klusteroiduissa järjestelmissä on otettava huomioon seuraavassa luvussa esiteltävän CAP-teoreeman tuomat rajoitukset ja NoSQL-tietokannoissa onkin otettu käyttöön BASE-käsite, joka on kevennetty vaihtoehto ACID-ominaisuuksille.

3.2 CAP-teoreema

Lyhenne CAP tulee sanoista eheys (engl. consistency), saatavuus (engl. availability) ja pirstoutumisen sietokyky (engl. partition tolerance). Hajautettu järjestelmä on eheä, jos kaikki sen solmut näkevät tiedot samaan aikaan. Saatavuus on hyvä, jos pystytään aina kirjoittamaan ja lukemaan tietoa. Pirstoutumisen sietokyky on kunnossa, jos järjestelmä toimii siitä huolimatta, että yhteys johonkin solmuun katkeaa. Hajautetussa järjestelmässä voi toimia ainoastaan kaksi CAP-teoreeman mukaisesta kolmesta ominaisuudesta. Tietokanta voi siis sisältää näistä ominaisuuksista parit eheys ja saatavuus, saatavuus ja pirstoutumisen sietokyky tai eheys ja pirstoutumisen sietokyky. [Brewer, 2000]

Järjestelmiin, joissa pirstoutumisen sietokyvystä luovutaan, kuuluvat kaikki perinteiset ACID-ominaisuuksia tukevat relaatiotietokannat. Brewer [2000] listaa tähän ryhmään myös LDAP-palvelimet ja xFS-tiedostojärjestelmän. Ominaisuutena tällaisilla järjestelmillä on yleensä kaksivaiheinen vahvistaminen ja välimuistin validointiprotokollat. ACID-ominaisuuksia tukemattomat NoSQL-tietokannat sijoittuvat pääsääntöisesti kahteen jälkimmäiseen kategoriaan. Järjestelmiin, joissa eheydestä luovutaan kuuluvat Brewerin [2000] mukaan hajautetut tietokannat, hajautettu lukitus ja enemmistöprotokollat (majority protocols). Ominaisuuksina tällaisilla järjestelmillä on pessimistinen lukitus ja vähemmistöosioiden saatavuuden estäminen. Järjestelmiin, joissa luovutaan saatavuudesta Brewerin [2000] mukaan kuuluvat Coda-tiedostojärjestelmä, web-välimuistipalvelut ja dns-palvelimet. Tyypillisiä piirteitä näille järjestelmille ovat vuokrauspalvelu ja vuokrien erääntymiset ja konfliktien ratkaiseminen. Hajautetuissa järjestelmissä joudutaan siis tekemään kompromisseja ja järjestelmät pitää valita sen mukaan, mikä sopii parhaiten käyttötarpeeseen.

3.3 BASE

BASE-lyhenne tulee sanoista periaatteessa käytettävissä (engl. basically available), ei aina oikeellinen (engl. soft state) ja lopulta eheä (engl. eventually consistent) [Pritchett, 2008]. Nämä ominaisuudet ovat joustavampi vaihtoehto jäykille ACID-ominaisuuksille. BASE-ominaisuuksien vaatimukset eivät ole yhtä tiukkoja kuin ACID-ominaisuuksissa, vaan oikeellisuutta ja eristyvyyttä on heikennetty. CAP-teoreemaa on usein käytetty perusteluna sille, miksi BASE-ominaisuuksia käytetään.

BASE-ominaisuuksia käytettäessä tietokannan ei tarvitse olla jokaisen transaktion jälkeen eheässä tilassa, vaan tiedot saattavat olla joissain tilanteissa vanhentuneita. Tietokannan tila palaa kuitenkin lopulta eheäksi. Nämä vaatimukset sopivat hyvin hajautetuille järjestelmille, jotka koostuvat usean tietokantapalvelimen klustereista. Tällaisen järjestelmän on hankala noudattaa täydellisiä ACID-ominaisuuksia, koska transaktiot ovat hajautettuja. BASE-ominaisuuksien asettamien vaatimusten

mukaan kaikkien solmujen ei tarvitse sisältää samaan aikaan kaikkia ajantasaisia tietoja koko ryppäästä, vaan solmujen tiedot päivittyvät kyllä ajan mittaan, joskin viiveellä.

3.4 NoSQL-tietomallit

Toisin kuin perinteisissä SQL-tietokannoissa, joissa tietokannan sisältö koostuu relaatioista ja niiden muodostamasta tietokantakaaviosta, NoSQL-tietokannoissa ei ole yhteistä mallia tiedon tallentamiselle. NoSQL-tietokannat luokitellaan useimmiten neljään kategoriaan: avain-arvo-varastoihin, dokumenttivarastoihin, sarakeperhevarastoihin ja graafitietokantoihin, jotka esitellään Hechtin ja Jablonskin [2011] käyttötapausorientoituneessa tutkimuksessa. Tähän tutkimukseen viitataan monissa yhteyksissä, kun NoSQL-tietokantoja luokitellaan. Graafitietokannoissa mielenkiintoisena piirteenä on se, että niiden tietomalli muistuttaa vanhoja verkkotietokantoja. Tässä tapauksessa on siis lähdetty kehittämään uudelleen vanhaa relaatiotietokantamallia edeltänyttä tietueista ja niiden välistä linkeistä koostunutta tietokantamallia. Markkinoilla olevat ratkaisut käyttävät jotakin näistä malleista tai ne saattavat tukea useampaa mallia.

3.4.1 Avain-arvo-varasto

Avain-arvo-varastot ovat tietokantoja, joissa on tallennettu arvoja ja niiden avain. Arvot ovat täysin näkymättömiä muulle järjestelmälle ja niihin päästään käsiksi vain tietämällä oikea avain. Arvot ovat keskenään riippumattomia ja niiden väliset riippuvuudet täytyy hallita niitä käyttävissä sovelluksissa. Koska tietokannan rakenne on näinkin yksinkertainen, tämän tyyppisessä tietokannassa ei ole ollenkaan tietokantakaaviota ja tämän ansiosta arvoja voidaan lisätä ja poistaa ilman, että tietokannan saatavuus häiriintyy. [Hecht and Jablonski, 2011]

Avain-arvo-varastojen etuna tavallisiin SQL-tietokantoihin nähden on niiden skaalautuvuus ja nopeus. Koska tietokantakaaviota ei ole ja tietokannan rakenne on varsin yksinkertainen, tietokanta skaalautuu hyvin useamman solmun klusterille. Kyselyistä tulee myös yksinkertaisia, koska käytettävä tietomalli on varsin yksinkertainen. Yksinkertaisimmillaan avain-arvo-varastoon tallennetaan vain avain-arvo-pari. Esimerkiksi avaimena voisi toimia ”kaupunki” ja sen arvona ”Helsinki”. Arvon Helsinki saa kysymällä avainta kaupunki.

Huonona puolena avain-arvo-varastoissa on se, että tietomallin yksinkertaisuudesta johtuen tietokantaan tallennettua tietoa ei voida käsitellä SQL-kielen tavoin suoraan tietokannassa, vaan varastoon tallennettujen arvojen yhdistely ja laskuoperaatiot on hoidettava ohjelmakoodissa. Avain-

arvo-varastot eivät tässä mielessä siis ole välttämättä hyvä valinta, jos data vaatii paljon käsittelyä, sillä tässä mielessä avain-arvo-varaston käyttö voi olla hankalaa ja hidasta.

Hyvänä esimerkkinä avain-arvo-varastosta on Redis, joka tätä työtä kirjoitettaessa on DB-Engines-sivustolla suosituin avain-arvo-varasto [Redis, 2015]. Rediksessä avaimen taakse voidaan tallentaa ihan vain yksinkertaisesti merkkijonoja, mutta myös merkkijonoista koostuvia listoja, joukkoja, järjestettyjä joukkoja ja hajautustauluja. Tiedon kyseleminen Rediksessä hoidetaan yksinkertaisesti kysymällä oikeaa avainta, jonka jälkeen Redis palauttaa avaimen takana olevan arvon. Muita suosittuja avain-arvo-varastoja ovat esimerkiksi Memcached [2009], Riak [2015], DynamoDB [2015], Aerospike [2015] ja Ehcache [2015] ja markkinoilla on myös kymmeniä muita toteutuksia, jotka on varustettu erilaisilla ominaisuuksilla.

3.4.2 Dokumenttivarasto

Dokumenttivarastoissa avain-arvo-parit on muodostettu siten, että avaimen arvona on yleensä JSON tai JSON:n kaltainen dokumenttirakenne. Dokumenttien sisällä avaimet tunnistavat koko dokumenttirakenteen ja näin jokaisella dokumenttirakenteella täytyy olla ainutlaatuinen tunniste. Toisin kuin avain-arvo-varastoissa, dokumenttivaraston arvot näkyvät myös muulle järjestelmälle ja dokumenttirakenteisiin voi kohdistaa myös kyselyitä. JSON-tyyppiset dokumenttirakenteet tukevat myös tietotyyppejä. Samoin kuin avain-arvo-varastoissa, dokumenttivarastoissa ei ole relaatiotietokannan kaltaista täsmällistä tietokantakaaviota. [Hecht and Jablonski, 2011]

JSON tulee sanoista JavaScript Object Notation. Se on Internet-sovelluksissa suosittu tietorakenne, jossa tieto on esitetty avain-arvo-muodossa. JSON:n tukemien tietotyyppejä ovat ainakin numerot, merkkijonot, totuusarvot, taulukot ja hakurakenteet [Bray, 2014]. Esimerkkinä JSON-muotoisesta dokumentista on seuraavalla sivulla esitetty tietorakenne.

```
{
   "etuNimi": "Erkki",
   "sukuNimi": "Mäkinen",
   "elossa": true,
   "ikä": 30,
   "pituus": 180,
   "osoite": {
      "katu": "Kemiankatu 8",
      "kaupunki": "Tampere",
      "postinumero": "33720"
   }
}
```

Useimmiten JSON-tyyppisiä dokumentteja sisältävä tietokanta koostuu kokoelmista, jotka sisältävät edellä kuvatun tyyppisiä dokumentteja. Kokoelmista voidaan hakea tietoa käyttäen SQL-kielistä tuttuja loogisia operaatioita. Esimerkiksi, jos meillä olisi kokoelma ihmiset ja siihen tehdään haku, jossa etunimi on ”Erkki” ja ikä on ”30”, niin kokoelma palauttaisi yllä kuvatun dokumentin.

Eräs hyvä esimerkki dokumenttivarastosta on MongoDB-tietokanta, joka on DB-Engines-sivustolta dokumenttivarastojen ensimmäisellä sijalla [MongoDB, 2015]. MongoDB tallentaa dokumentin BSON-formaatissa, joka on eräänlainen laajennus JSON-formaattiin. BSON tulee sanoista Binary JSON ja BSON dokumentit ovatkin JSON-dokumentteja binäärimuodossa. BSON laajentaa JSON-formaattiin päivämäärätyypin ja binääridatatyypin [BSON, 2014]. BSON-formaatilla antaa tietyissä tilanteissa etuja JSON-formaattiin nähden. BSON:n ominaisuuksissa on panostettu keveyteen, käännettävyyteen ja tehokkuuteen.

3.4.3 Sarakeperhevarasto

Sarakeperhevarastojen esikuvana on Googlen käyttämä Bigtable [Chang et al., 2006]. BigTablen tietomallia kutsutaan moniulotteiseksi järjestetyksi kartaksi. Sarakeperhevarastojen perusyksikkönä on sarake, joka koostuu sarakeavaimesta ja sen alla olevasta arvosta. Sarakkeista muodostetetaan rivejä antamalla sarakejoukolle riviavain. Lisäksi voi olla myös supersarakkeita, jotka sisältävät useita

alisarakkeita. Näiden avulla voidaan muodostaa sarakehierarkkioita ja niitä voi myös käyttää apuna, kun halutaan luoda relaatioita.

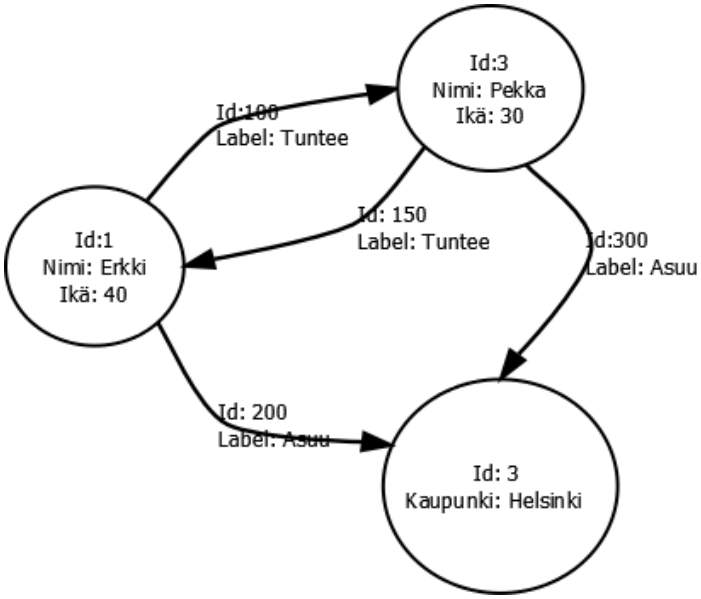
Sarakeperhevarastot tuovat perinteisiin tietokantoihin nähden etuna lisää suorituskykyä, mikäli haettava tieto on helposti tunnistettavissa. Perinteisissä relaatiotietokannoissa tiedot tallennetaan rivipohjaisesti. Jos tällaisessa tietokannassa halutaan hakea jokin tietty arvojoukko, joudutaan käsittelemään usein myös paljon muuta epäolennaista informaatiota. Kun haetaan jotakin tiettyä monikkoa, joudutaan mahdollisesti vertailemaan useita kenttiä ja lisäksi mahdollisesti tehdään taululiitoksia. Sarakeperhevarastoissa voidaan vain hakea koko monikko yksinkertaisesti kutsumalla oikeaa sarakeavainta.

DB-Engines [2015b] sivuston mukaan tätä työtä kirjoitettaessa suosituimpana sarakeperhevarastona on Apachen Cassandra. Cassandran käyttö muistuttaa perinteisiä SQL-tietokantoja. Kaavioiden sijasta Cassandrassa luodaan avainavaruus (engl. key space). Avainavaruuteen luodaan sarakeperheitä ja näistä voi hakea dataa Cassandran omalla SQL-kieltä muistuttavalla CQL-kielellä. Liitoslauseita kyselyissä ei tosin voi tehdä, koska Cassandra ei tue relaatioita. Muita merkittäviä sarakeperhevarastoja ovat Accumulo [2015], HBase [2015] ja Hypertable [2015].

3.4.4 Graafitietokanta

Graafitietokantojen tietomalli perustuu graafiteoriaan, joka on matematiikan osa-alue. Näissä tietokannoissa käsitteet, kuten henkilöt, esineet ja kohteet kuvataan solmujen (engl. nodes) avulla ja käsitteiden väliset suhteet kuvataan kaarilla (engl. edges) [Wiggins, 2010]. Nämä tietokannat ovat hyviä datalle, jossa käsitellään paljon kohteiden välisiä suhteita. Tällaista dataa on esimerkiksi ihmisten väliset suhteet sosiaalisissa verkostoissa ja linkit web-sivujen välillä.

Graafitietokannat ovat perinteisiä SQL-tietokantoja tehokkaampia paljon erilaisia suhteita sisältävän datan käsittelyssä. Tavalliset SQL-tietokannat ovat varsin kömpelöitä, mikäli suhteita tulee paljon joidenkin taulujen välille. Tällöin tarvitaan suhdetaulu, jonka kautta useampi suhde hallitaan. Tämä lisää taulujen määrä tietokannassa. Myös SQL-kyselyistä tulee varsin pitkiä, kun monia tauluja yhdistetään liitoslauseiden avulla [Wiggins, 2010]. Graafitietokannoissa suhdetiedot voidaan hakea suoraan solmuista ja erillistä relaatiotaulua ei tarvita. Kyselyistäkin saadaan näin ollen yksinkertaisempia ja tämä nopeuttaa suhteiden käsittelyä.



Kuva 2: Graafitietokannan tietomalli.

Kuvassa 2 nähdään yksinkertaistettu esimerkki graafitietokannan rakenteesta. Tietueet esitetään kolmena solmuna, jotka tässä tapauksessa ovat kaksi ihmistä ja kaupunki. Tietueiden välisiä relaatioita ilmaisevat solmujen väliset kaaret, jotka tässä tapauksessa ilmaisevat ihmisten väliset suhteet toisiinsa ja missä kaupungissa he asuvat.

DB-Engines-sivuston mukaan selkeästi suosituin graafitietokanta tällä hetkellä on Neo4J [2014]. Kuten aikaisemmin mainittiin, tietokannan tietosisältö tallennetaan solmuihin ja niiden välille määritetään relaatioita. Tästä koostuu Neo4J:n tietomalli. Tietokannassa on oma kyselykieli Cypher, joka muistuttaa SQL-kyselykieltä. Kielen avulla voidaan luoda tietokantaan solmuja, viedä niihin dataa ja määrittää niiden välille relaatioita.

3.5 NewSQL-tietokannat

Käsitteellä NewSQL viitataan yleensä tietokantoihin, joissa yhdistyy perinteisten SQL-tietokantojen ACID-eheysvaatimukset ja NoSQL-tietokantojen hyvä skaalautuvuus. Käsitteenä NewSQL on varsin uusi. Käsitettä käytti ensimmäisenä Matt Aslett [2011] artikkelissaan ”How will the database incumbents respond to NoSQL and NewSQL?”. Monissa ratkaisuissa on tarvetta perinteisten relaatiotietokantojen tarjoamalle eheydelle, mutta dataa on silti niin paljon, että tarvitaan lisäksi hyvää skaalautuvuutta. Tällöin uusista tietokantakaaviottomista NoSQL-ratkaisuista ei ole hyötyä.

Venkateshin [2012] mukaan markkinoilla olevilla NewSQL-ratkaisuilla on ainakin viisi yhteistä piirrettä. Kaikissa käytetään SQL-kieltä ensisijaisena kyselykielenä; transaktioissa tuetaan ACID-mallia; lukoton rinnakkaisuuden hallinta siten, että lukeminen ei ole konfliktissa kirjoituksen kanssa; arkkitehtuurit tarjoavat parempaa solmukohtaista suorituskykyä kuin perinteiset relaatiotietokannat; hyvin skaalautuva shared nothing -arkkitehtuuri, jonka avulla tietokannan kuorma jakautuu tasaisesti.

Venkateshin [2012] mukaan NewSQL-tietokantoja voidaan kategorisoida kolmeen ryhmään. Ensimmäinen ryhmä ovat täysin uudet tietokannat, jotka sisältävät puhtaalta pöydältä kirjoitettua ohjelmakoodia. Sovellukset voivat olla pelkästään ohjelmallisia tai laitteistotuettuja. Toinen ryhmä ovat MySQL-tietokantaan pohjautuvat järjestelmät. Näissä sovelluksissa on kaikki MySQL:n ominaisuudet, mutta tämän lisäksi niissä on kehitetty tietokannan skaalautuvuutta useammalle solmulle. Kolmas ryhmä on läpinäkyvä klusterointi (engl. transparent clustering). Tällaisissa ratkaisuissa NewSQL-toteutus toimii kerroksena jonkin perinteisen SQL-tietokannan ja sovelluksen välissä tarjoten mahdollisuuden skaalata tietokanta usealle solmulle.

NewSQL-tietokantojen joukko on tätä tutkimusta tehdessä paljon NoSQL-tietokantoja pienempi. Www.nosql-database.org-sivusto listaa näitä vajaa kaksikymmentä. DB-Ranking listaa NewSQL tietokannat perinteisten tietokantojen joukossa. Venkateshin [2012] luokittelemiin ensimmäisen ryhmän sovelluksiin kuuluvat ainakin VoltDB [2015], MemSQL [2015] ja NuoDB [2015]. Toiseen ryhmään kuuluvat TokuDB [2015] ja InfiniDB [2015], GenieDB [2015], JustOneDB [2015] ja kolmanteen ryhmään kuuluvat dbShards [2015], Scalearc [2015] ja ScaleBase [2015]. NewSQL-tietokannat ovat vielä kehittymisvaiheessa ja mitään asemaansa vakiinnuttaneita tuotteita ei tätä tutkimusta tehdessä ole.

3.6 Rajapinnat ja kyselykieli

NoSQL-tietokannoissa ei ole yhtenäistä rajapintaa, kuten relaatiotietokantojen ODBC ja JDBC. DB-ranking-sivuston mukaan NoSQL-tietokannoista JDBC:tä tukevat vain Sqrrl [2015] ja XAP [2015]. Näistä Sqrrl tukee kaikkia neljää tietomallia ja XAP on avain-arvo-tietokanta. ODBC:tä tukevat Sqrrl [2015] ja MarkLogic [2015], joka on dokumenttivarasto. Joissakin tapauksissa voidaaan käyttää erillisen kolmannen osapuolen tarjoamia JDBC- ja ODBC-kirjastoja, joille tietokannan valmistaja ei tarjoa virallista tukea. SQL-kieltä erillisen tietokannan päälle liitettävän kerroksen kautta tukevat BerkeleyDB [2015] (avain-arvo-varasto) ja FoundationDB [2015] (avain-arvo-varasto, dokumenttivarasto ja relaatiotietokanta).

Koska standardoitua rajapintaa ei ole, tietokantayhteys on toteutettu eri toteutuksissa erilaisilla sovelluskohtaisilla ratkaisuilla. Syynä tähän on se, että näille tietokannoille ei löydy ainakaan toistaiseksi mitään asemansa vakiinnuttanutta yleistä kyselykieltä. Yritystä tällaisen kehittämiseksi kuitenkin on. Yleistä NoSQL-kyselykieltä on pyritty kehittämään UnQL-kielestä [Couchbase, 2011]. NewSQL-tietokantaratkaisuissa pääsääntöisesti tuetaan ODBC- ja JDBC-rajapintoja, sillä niistä pyritään tekemään SQL-yhteensopivia.

Monissa NoSQL-tietokannoissa käytetään Googlen kehittämää MapReduce-algoritmia datan ryhmittelyyn [Dean and Ghemawat, 2008]. Algoritmissa on kaksi osaa Map ja Reduce. Esimerkiksi saman avaimen alle menevien arvojen määrän laskeminen voidaan toteuttaa siten, että Map ottaa sisäänsä avain-arvo-pareja ja tuottaa väliaikaisen arvojoukon. Arvojoukko on ryhmitelty samanlaisten avainten mukaan. Reduce hoitaa samannimisten avainten alla olevien arvojen määrän laskemisen ja antaa ulostulona avaimen, arvon ja arvojen määrän saman avaimen alla. Map- ja Reduce-funktioita voidaan ajaa rinnakkain ja näin saadaan hyödynnettyä esimerkiksi palvelinklusterin koko laskentateho.

3.7 Tietotyypit

NoSQL-tietokannoissa käytetyt tietotyypit vaihtelevat toteutuskohtaisesti. Avain-arvo-kannoissa data tallennetaan usein vain binäärimuodossa, merkkijonoja tai jonakin tietorakenteena kuten assosiaatiotaulu, joukko, lista, puu tai luokka. Dokumenttivarastoissa data tallennetaan JSON-muodossa, joten niissä käytetään JSON:n tietotyyppejä number, string, boolean, taulukko, olio ja null [JSON, 2014]. Graafitietokannoissa tietotyypeillä on myös vaihtelua avain-arvo-varastojen tapaan, mutta pääsääntöisesti monet tyypitetyissä ohjelmointikielissä tuetut datatyypit ovat tuettuja. Esimerkiksi Neo4j:n tukemia tietotyyppejä ovat byte, short, int, long, float, double, char ja string. Sarakeperhevarastoissa on samanlainen esimerkiksi Cassandran CQL-kielestä löytyy tietotyypit merkkijonoille, kokonaisluvuille, totuusarvoille ja aikaleimoille.

4 Verkkotapahtumien keräysjärjestelmä

4.1 Yritys

Kyseessä oleva yritys on kansainvälinen tietoliikennelaitteita valmistava yhtiö, jolla on jo alalta useamman vuoden kokemus. Yrityksen tuotteet ovat kansainvälisesti käytössä useassa maassa eri operaattoreilla ympäri maailmaa. Yrityksellä on teleliikennelaitteiden valmistamisesta useiden vuosien kokemus ja yrityksen tuotteet ovat tunnetusti kilpailukykyisiä ja arvostettuja.

4.2 Yleiskuvaus

Matkapuhelinverkkojen kehityksen myötä verkkojen koko on kasvanut varsin suureksi ja liikenteen määrä kasvanut huomattavasti. Lisäksi monessa verkossa käytetään useita eri teknologioita. Samoissa verkoissa voidaan käyttää sekä 2G (GSM, GPRS), 3G (W-CDMA, HSPA, UMTS), 4G (WiMAX, LTE), että IP-multimediaa. Kun liikenteen määrä kasvaa ja verkon rakenne suurenee, tulee tarpeelliseksi keskittyä palvelun laadun varmistamiseen.

Ratkaisuna laadun varmistamiseen on tutkimuksessa kyseessä olevassa yrityksessä kehitetty verkkotapahtumien keräysjärjestelmä. Tämän järjestelmän avulla voidaan monitoroida matkapuhelinverkon tapahtumia. Tapahtumia voivat olla niin puheluiden ja tekstiviestien aiheuttamat verkkotapahtumat kuin pakettipohjaisen datan, Internet-liikenteen, aiheuttamat verkkotapahtumat. Tapahtumia voidaan seurata reaaliajassa järjestelmän avulla ja järjestelmä osaa muodostaa niistä visuaalisia esityksiä.

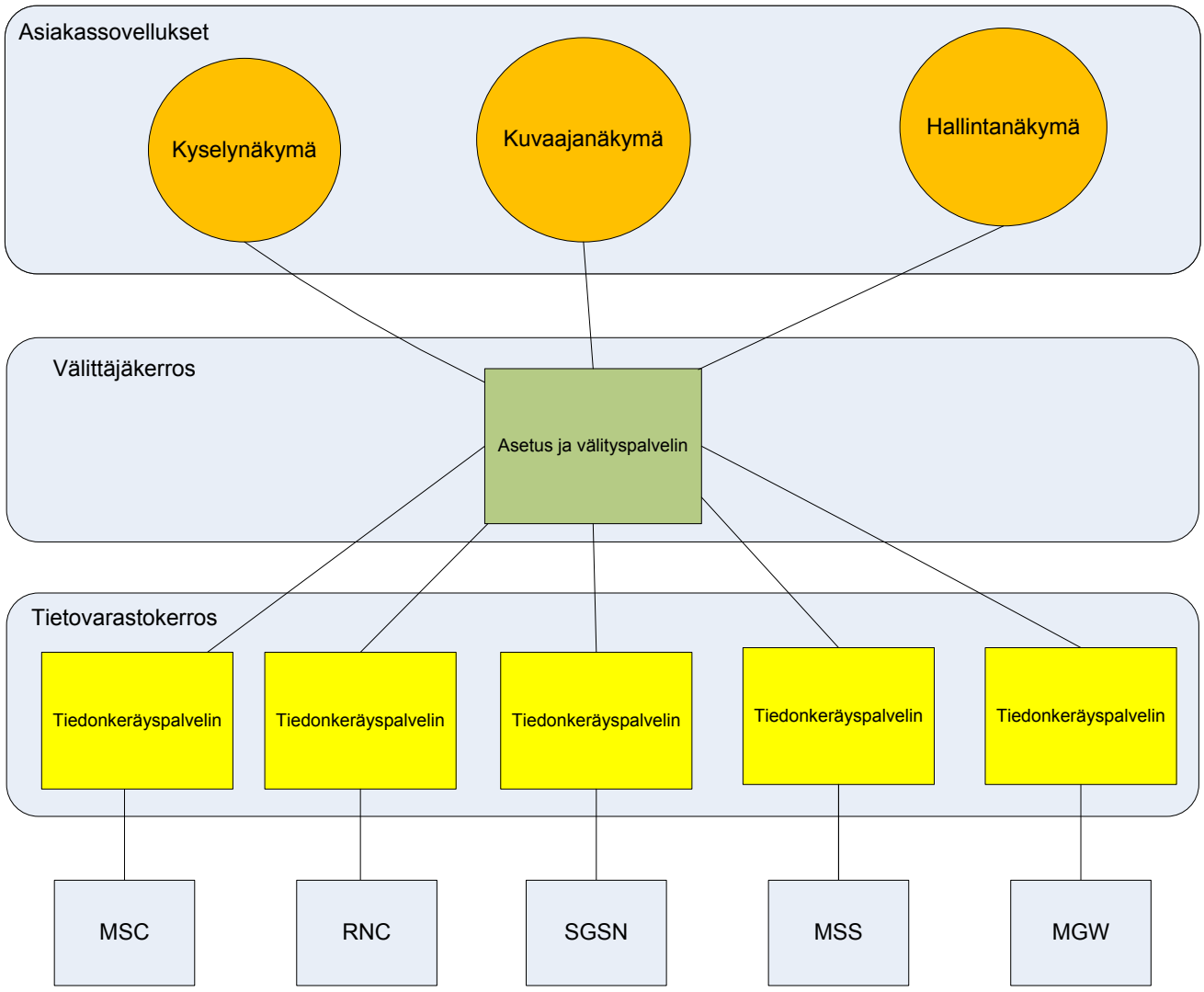
Järjestelmän avulla voidaan paitsi havainnoida tietoliikenteessä tapahtuneita virheitä, myös tarkkailla verkon sisältämiä laitteita ja yhteyskäyttäytymistä. Esimerkiksi asiakkaiden käyttämien matkapuhelinten mallit saadaan selville järjestelmään tulevien raporttien tiedoista ja tämän avulla voidaan tarkkailla verkon toimivuutta eri puhelinmalleilla. Järjestelmän avulla saadaan selville verkosta myös mahdolliset luvattomat puhelut. Voidaan esimerkiksi huomata sellaiset tilanteet, jolloin yleensä vähän käytetyllä puhelimella soitetaankin yhtäkkiä pitkiä puheluita numeroihin, joihin sillä ei ole yleensä soitettu [Arvonen, 2003].

Verkkotapahtumien keräysjärjestelmän liiketoiminnallisina hyötyinä on ylläpidon parantuminen ja sitä kautta operointikustannusten (OPEX) väheneminen. Myös pääomakustannukset (CAPEX) vähenevät, sillä järjestelmä tarjoaa suoran käyttöliittymän verkkoelementteihin ja näin säästetään elementtien

huoltoon liittyvissä menoissa. Järjestelmä parantaa myös asiakastyytyväisyyttä ja sitä kautta verkon käyttö ja tuottavuus kasvavat.

4.3 Arkkitehtuuri

Järjestelmässä käytetään multi tier -arkkitehtuuria. Pääsääntöisesti järjestelmä voidaan jakaa neljään kerrokseen, joista alimmassa kerroksessa sijaitsevat elementit, toisessa kerroksessa tiedonkeräyspalvelimet, kolmannessa kerroksessa välittäjäpalvelimet ja ylimmässä kerroksessa asiakassovellukset.



Kuva 3: Verkkotapahtumien keräysjärjestelmän arkkitehtuuri.

Kuvassa 3 nähdään, verkkotapahtumien keräysjärjestelmän arkkitehtuuri. Alimmalla tasolla sijaitsevat verkkoelementit. Niitä on useaa tyyppiä ja elementti voi olla esimerkiksi puhelinkeskus, joka vastaa puhelujen yhdistämisestä. Järjestelmän tukemia elementtityyppejä ovat 3G-verkoissa yleisesti käytössä olevat verkkoelementit FlexiNS, FlexiNG, SGSN, DX-HLR, BSC, RNC, EnodeB, OpenTAS, MSC/MSS, MGW, OpenBGW, CFX-5000, joista saadaan kerättyä dataa järjestelmään.

Toisella tasolla sijaitsevat tiedonkerääjäpalvelimet. Nämä palvelimet vastaanottavat elementeiltä tulevia RTT-raportteja (engl. Real Time Traffic). Tällainen raportti tehdään aina, kun verkossa esiintyy jokin verkkotapahtuma. Kaikki raportit tallennetaan tiedonkerääjäpalvelimien tietokantaan. Lisäksi, kun raportti saapuu, siitä lasketaan tunnuslukuja (engl. key performance indicator). Tunnuslukuja voivat olla esimerkiksi onnistuneiden soittojen määrä ja vastattujen puheluiden määrä. Tunnuslukujen laskeminen on varsin hyödyllistä, sillä tietokanta saattaa kasvaa varsin suureksi, jos liikennettä on paljon. Tällöin tunnusluvut auttavat hahmottamaan yleiskuvaa verkon tilasta.

Kolmannella tasolla sijaitsevat välittäjäpalvelimet. Näiden tärkeimpänä tehtävänä on toimia välikerroksena toisen ja neljännen kerroksen välillä. Välittäjäpalvelimissa voi olla kiinni useita tiedonkerääjäpalvelimia. Järjestelmässä saattaa olla kahdenlaisia eri kokoonpanoratkaisuja tiedonkeräyspalvelimien suhteen. Perusratkaisussa käytössä on vain yksi välittäjäpalvelin ja kaikki tiedonkeräyspalvelimet on kytketty suoraan kiinni tähän palvelimeen. Jos tiedonkeräyspalvelimia on paljon, voidaan käyttää useampaa tiedonkeräyspalvelinta jakamaan tiedonkeräyspalvelinten aiheuttamaa kuormaa. Tämän lisäksi on olemassa ratkaisu, jossa tiedonkeräyspalvelimet sijaitsevat kahdella tasolla.

Neljäs taso on asiakastaso. Tällä tasolla sijaitsevat kaikki sovellukset, jotka käyttävät välittäjäpalvelimien sisältäviä palveluja. Nämä sovellukset tarjoavat loppukäyttäjälle pääsyn välittäjäpalvelimien kautta tiedonkerääjäpalvelimien dataan. Sovelluksien kautta käyttäjä pääsee esimerkiksi selaamaan tiedon keräyspalvelimille kerättyjä raportteja, tarkastelemaan raporteista tehtyjä kuvaajia ja muuttamaan järjestelmään liittyviä asetuksia.

Fyysisesti ensimmäisen ja toisen tason laitteet sijaitsevat yleensä samoissa tiloissa. Jokaista verkkoelementtiä kohden on yksi tiedonkeräyspalvelin ja ne sijoitetaan yleensä lähelle toisiaan. Yleensä tiedonkeräyspalvelimet ja verkkoelementit sijaitsevat operaattorien datakeskuksissa. Välittäjäpalvelimet sijaitsevat taas yleensä operaattorin valvomon läheisyydessä. Asiakassovelluksia

käyttävät loppukäyttäjät ovat yleensä lähellä välittäjäpalvelimia, mutta tämä ei ole yleinen vaatimus. [Mäki, 2008]

4.4 Tietokanta ja tiedon vienti tietokantaan

Elementtien lähettämät raportit kerätään tiedonkeräyspalvelimien tietokantaan. Tietokannassa ei ole yhtenäistä tietokantakaaviota, vaan jokaiselle raporttityypille on oma taulunsa. Raporttien sisältö vaihtelee elementtikohtaisesti ja eri tiedonkeräyspalvelinten tietokannoissa on omanlaisensa joukko tauluja elementtityypistä riippuen.

Elementeiltä saadaan tietoa kahdella eri tavalla. Se voidaan kerätä joko suoraan elementeiltä tai elementtien välille kiinnitettyjen antureiden avulla. Suoraa tiedonkeräysmallia voidaan käyttää yrityksen omien elementtien kanssa. Omissa elementeissä elementtien rajapinnat tunnetaan, joten tässä tapauksessa suora tiedonkeräys on mahdollista. Toisten toimittajien elementtien rajapinnat ovat suljettuja, joten tiedon keräys on tehtävä elementtien ulkopuolelta anturien avulla.

Fyysisesti tietokanta sijaitsee aina tiedonkeräyspalvelimella. Koska järjestelmässä on useita tiedonkeräyspalvelimia, järjestelmässä on myös useita tietokantoja. Jokaista elementtiä kohden järjestelmässä on yksi tietokanta elementin raporteille. Myös välittäjäkerroksen palvelimelle voidaan asentaa tietokanta tarvittaessa, mutta tämä on täysin vapaaehtoista. Välittäjäkerroksessa toimivat sovellukset on suunniteltu siten, että ne osaavat hakea tarvittavan datan välittäjäpalvelimeen kiinnitetyn tiedonkeräyspalvelimen tietokannasta.

4.5 Tietokannan vaatimukset

4.5.1 Lainsäädännölliset vaatimukset

Verkkotapahtumien keräysjärjestelmässä käytettäviin tietokantoihin tallennetaan paljon yksityisten asiakkaiden arkaluontoisia tietoja. Järjestelmään tallennetaan muun muassa matkapuhelinkohtaiset IMSI- ja IMEI-tiedot, joiden avulla voidaan tunnistaa puhelin ja puhelimen SIM-kortti. Lisäksi asiakkaiden puhelu- ja tekstiviestikäyttäytymisestä tallennetaan paljon tietoa, joiden perusteella voidaan saada tietoa yhteydenottotapahtumasta. Näistä syistä tietojen tallentamisessa pitää ottaa huomioon yksityisyyden suojaa koskeva lainsäädäntö. Tässä luvussa käsitellään Suomen ja Euroopan unionin laeissa ja direktiiveissä esitetyt kohdat siten kuin ne koskevat verkkotapahtumien keräysjärjestelmää.

Suomen laissa teletunnistetietojen käsittelyä säätelee tietoyhteiskuntakaari [2014/917, 2014]. Tämä laki astui voimaan 1.1.2015 ja korvasi aikaisemmin voimassa olleen sähköisen viestinnän tietosuojalain. Tietoyhteiskuntakaaressa verkkotapahtumien keräysjärjestelmää koskee erityisesti luku 17, jossa käsitellään sähköisen viestin ja välitystietojen käsittelyä. Tämän luvun alla pykälä 137 määrittää yleiset käsittelyperiaatteet viestinnän välittäjälle. Välitystietojen luovutus on rajoitettu ainoastaan niille tahoille, keillä on oikeus käsitellä tietoja asianomaisessa tilanteessa. Kun välitystiedot on käsitelty, tiedot pitää hävittää. Tietoja saa käsitellä vain viestinnän välittäjä tai tilaajan lukuun toimiva taho, joka käsittelee viestejä ja välitystietoja laissa säädettyjen tarkoitusten toteuttamiseksi. Verkkotapahtumien keräysjärjestelmässä näitä tarkoituksia ovat 141 § Käsittely teknistä toteuttamista varten, 142 § Käsittely tilastollista analyysia varten ja 144 § Käsittely teknisen vian tai virheen havaitsemiseksi. Näiden lisäksi 145 § säätää käsittelyä koskevien tietojen tallentamisesta. Tässä velvoitetaan säilyttämään yksityiskohtaiset tapahtumatiedot välitystietojen käsittelystä, mikäli se on teknisesti ja ilman kohtuuttomia kustannuksia mahdollista. Tietoja tulee lain mukaan säilyttää kaksi vuotta niiden tallentamisesta.

Euroopan unionin tasolla järjestelmän käsittelemiä tietoja koskee Euroopan parlamentin ja neuvoston direktiivi [2002/58/EY, 2002], eli sähköisen viestinnän tietosuojadirektiivi. Tässä on kyseessä sähköisen viestinnän tietosuojadirektiivi, jota sovelletaan koko Euroopan unionin alueella. Tässä direktiivissä verkkotapahtumien keräysjärjestelmää koskee erityisesti kuudes artikla, jossa määrätään liikennetiedoista. Tämän artiklan ensimmäisessä kohdassa sanotaan, että tilaajia ja käyttäjiä koskevat liikennetiedot, joita palveluntarjoaja käsittelee, pitää poistaa heti kun niitä ei enää tarvita viestinnän välittämiseen. Kuitenkin siten, että artiklan kohtia 2, 3 ja 5 ei rajoiteta.

Kohdista 2, 3 ja 5 kohta viisi koskee verkkotapahtumien keräysjärjestelmää. Kohta viisi rajoittaa liikennetietojen käsittelyn ainoastaan niiden palvelua tarjoavien vastuulla toimiviin henkilöihin, joiden toimenkuvaan kuuluu laskutuksen, liikenteenhallinnan, asiakastiedustelujen, petosten paljastusten ja sähköisten viestintäpalvelujen markkinoinnin tai lisäarvopalvelujen tarjoaminen. Liikennetietojen käsittelyä saa harjoittaa vain kyseisten toimien vaatimassa laajuudessa.

Muilta osin sähköisen viestinnän tietosuojadirektiivi ei sisällä verkkotapahtumien keräysjärjestelmää koskevia kohtia. Suurin osa järjestelmää koskevista lakipykälistä löytyy siis Suomen laista. Tämän lisäksi jokaisessa maassa on omat lainsäädännölliset rajoituksensa, mutta tässä tutkimuksessa keskitytään Suomen ja Euroopan Unionin lakiin. Näistä oikeuksista ja rajoituksista seuraa joukko vaatimuksia, joita verkkotapahtumien keräysjärjestelmän tietokannan tulee täyttää.

Koska tunnistetietojen käsittely on rajattu vain tiettyjen henkilöiden vastuulle, tietokantaan pääsy on pystyttävä rajaamaan vain tietyille käyttäjille. Tietokannan on oltava turvallinen siten, että tietosisältö suojataan käyttäjätunnuksien taakse. Tiedot on myös pystyttävä hävittämään vaaditun ajan kuluessa. Viidestoista pykälä asettaa vaatimukset myös tunnistetietojen käsittelyyn liittyvien tietojen tallentamiselle. On pystyttävä tallentamaan tiedot siitä kuka on käynyt tietokannassa ja mihin aikaan. Tiedot on pystyttävä säilyttämään vähintään kaksi vuotta niiden tallentamisesta. Näiden lisäksi tietokannan on oltava sellainen, että tietoturvasta pystytään huolehtimaan viidennen luvun lakien vaatimalla tavalla.

Turvallisuuden takaamiseksi suunnittelussa on aina lähdettävä siitä, että henkilöihin, jotka pääsevät järjestelmän dataa käsittelemään, ei voi luottaa. Sellaista tilannetta ei saa syntyä, että operaattorin työntekijä ottaa tietokannasta kopion ja vie sen kotiin. Suora pääsy tietokantaan on estettävä ja tietokantaan pääsy on hoidettava vain siihen tarkoitetuilla sovelluksilla. Tietokannan tietosisältö ei saisi mielellään olla helposti avattavissa, jos joku saa sen käsiinsä. Tiedostot, joihin tietokanta tallentaa tietokannan sisällön, eivät saisi olla helposti tutkittavissa ja avattavissa.

4.5.2 Tekniset vaatimukset

Tekniset vaatimukset tietokannalle asettavat paitsi lainsäädäntö, verkkotapahtumien keräysjärjestelmässä tehdyt tekniset ratkaisut, myös verkossa liikkuvan datan määrä. Nykyisessä toteutuksessa tietokanta on integroitu osaksi sovellusta ja sovellus tekee tietokantaan hakuja SQL-kielellä. Asiakkaalle tietokanta ei näy suoraan, vaan asiakas käyttää sitä sovelluksen kautta. Muutenkin tietoturvan kannalta tietokantaan ei saisi olla suoraa pääsyä. Nykyinen käytössä oleva tietokanta on perinteinen SQL-tietokanta. Näistä mainituista asioista seuraa ainakin seuraavia vaatimuksia:

- Tietokannan pitää toimia itsenäisesti ilman, että tarvitaan ylläpitotoimia.
- Jos edellinen ei ole mahdollista, pitäisi olla mahdollisuus rajata ylläpitäjän oikeuksia siten, että ylläpitäjän ei ole mahdollista kysellä tai viedä tallennettua dataa tai luoda käyttäjiä, joilla on tähän oikeuksia.
- Mahdollisuus kryptata kanta saraketasolla.
- Mahdollisuus täydelliseen tietokannan kryptaukseen.
- Tietokannan käyttäminen SQL-kielellä.
- Tuki ODBC:lle, JDBC:lle ja/tai C-kielisille ja Java-kielisille rajapinnoille.

- Tuki numeerisille datatyypeille tinyint, smallint, int, bigint, float, double ja mielellään myös etumerkittömille (eng. unsigned) luvuille.
- Tuki merkkityypille (CHAR) ja merkkijonotyypille (VARCHAR) ja Unicodelle.
- Tuki binäärityypeille BINARY ja VARBINARY.
- Tuki aikaa kuvaaville tyypeille DATE, TIME, TIMESTAMP ja karkaussekunneille.
- Aikaleiman tarkkuus pitäisi olla vähintään millisekunneissa tai parempi.
- Tietokannassa olisi hyvä olla mahdollisuus klusteroida tietokanta siten, että jokaiselle tietokantasolmulle voidaan määrittää, mitä dataa se sisältää.
- Tietokanta pitäisi pystyä hajauttamaan fyysisesti siten, että tietokannan osat voisivat olla maantieteellisesti eri paikoissa. Näin tietokanta sopisi hyvin verkkotapahtumien keräysjärjestelmän arkkitehtuuriin.
- Mahdollisuus muuttaa tietokannan sisältöä dynaamisesti lisäämällä ja poistamalla sarakkeita, muuttamalla indeksointia ja lisäämällä uusia tauluja.
- Mahdollisuus säätää datan poistaminen tietokannasta siten, että arkaluontoiset tiedot voidaan poistaa lain asettamassa ajassa ja tilanne, jossa kiintolevy täyttyy, voidaan estää.
- Automaattinen käynnistys, jos tietokanta kaatuu.
- Mahdollisuus transaktiolokeille.
- Vakaa 250 000 riviä sekunnissa sisäänvienti yhdellä HP DL360 Gen9 laitteistoon perustuvalla tietokantasolmulla, jotta järjestelmä hallitsee suuren dataliikenteen aiheuttaman kuorman.
- Mahdollisuus viedä nykyisessä tietokannassa olevaa sisältöä ja käyttää tietokannassa käytettyjä tietotyyppejä.
- Skaalautuvuus petatavun tallennuskapasiteettiin saakka.
- Tuki VM-Warelle ja Open Stackille ja näiden pilviversioille.

4.6 Nykyinen toteutus vaatimusten kannalta

4.6.1 Ratkaisut lainsäädännöllisten vaatimusten toteuttamiseksi

Nykyisessä järjestelmässä vaatimukset tietoihin pääsyn rajauksista on toteutettu siten, että tietokantaan pääsy on rajattu vain järjestelmän omille asiakasohjelmille. Ohjelmien käyttäjätunnuksia hallinnoidaan käyttäjä- ja konfiguraatiotiedon hallintajärjestelmän avulla. Tietokantaan pääsy muilla kuin järjestelmän omilla asiakassovelluksilla, on oletusarvoisesti kielletty ja sen voi tarvittaessa sallia vain järjestelmän omalla hallintaohjelmalla. Tietokanta on suljettua lähdekoodia ja ne määritykset, kuinka tietokanta tallentaa kannan, eivät ole yleisesti tiedossa. Tästä johtuen tiedon tallennustapaa on vaikea takaisinmallintaa ja tietokantatiedostot saa selkokielisenä auki vain tietokantaohjelmistolla.

Tunnistetietojen säilytystä tietokannassa voidaan hoitaa tapauskohtaisesti. Tietojen poisto hoidetaan automatisoidusti ja poistokriteerit voidaan säätää tallennustilan määrän ja ajan mukaan rekisteriavaimien arvoja muuttamalla. Näin voidaan vastata EU:n, Suomen ja muiden maiden vaatimuksiin siitä, että tiedot tulee hävittää, kun niitä ei enää tarvita käsittelyä varten. Tulevaisuudessa tietokantatiedostot voitaisiin salata suojausavaimella. Myös koko kiintolevy, jossa tietokanta sijaitsee, voitaisiin kryptata. Haittapuolena kryptauksessa on, että se saattaa hidastaa merkittävästi tietokannan toimintaa.

4.6.2 Tekniset ominaisuudet

Nykyinen käytössä oleva tietokanta on tyypiltään perinteinen SQL-yhteensopiva ja ACID-ominaisuudet täyttävä relaatiotietokanta. Tietokannan data on siis tallennettu tauluihin ja dataa voidaan kysellä ja viedä sisään ODBC- ja JDBC-rajapintojen kautta. Tietokanta on sulautettu tietokanta (eng. embedded database) ja tämä mahdollistaa ohjelman sulauttamisen osaksi verkkotapahtumien keräysjärjestelmää siten, että kantaa voi käyttää vain järjestelmän avulla. Näin saadaan ratkaistua lainsäädännön määrittelemät turvallisuusvaatimukset. Toinen hyvä puoli tässä ratkaisussa on, että tietokanta ei tarvitse ylläpitoa (eng. zero administration) vaan verkkotapahtumien keräysjärjestelmä hoitaa tietokannan käsittelemisen automaattisesti. Tapa, jolla tietokanta tallentaa datan ei ole yleisessä tiedossa ja tietokannan datatiedostoissa oleva data ei näy selkokielisenä käyttäjälle, jos niitä yrittää avata tekstieditorissa. Tietokannassa on tuki kannan käytön auditoinnille. Verkkotapahtumien keräysjärjestelmä kuitenkin käyttää tietokantaa siten, että järjestelmän käyttäjätiedot eivät näy tietokannalle, joten tästä syystä tietokannan auditointia ei käytetä. Auditointiominaisuus on toteutettu verkkotapahtumien keräysjärjestelmässä lainsäädännöllisten vaatimusten täyttämiseksi. Myös datan automaattinen poisto on toteutettu verkkotapahtumien keräysjärjestelmässä ja ajan pystyy

määrittämään tapauskohtaisesti verkkotapahtumien keräysjärjestelmän asetuksista. Tietokannan suorituskyky on HP ProLiant DL380 Gen9 Server -palvelinkoneella 210 000 raporttia sisään sekunnissa, kun raporttia syöttäviä säikeitä on 16.

5 Tietokantojen tutkiminen

5.1 Aikaisemmat tutkimukset

NoSQL-tietokannoista löytyy varsin paljon erilaisia tutkimusartikkeleita, joissa on vertailtu tietokantojen ominaisuuksia. Eri tutkimuksissa on tutkittu vaihtelevaa joukkoa erilaisia markkinoilla olevia tietokantoja. Monissakaan tutkimuksissa ei ole tehty mitään kaikkia NoSQL-tietokantoja kattavaa vertailua, vaan useimmissa on vertailtu yleisimmin käytettyjä tietokantoja tai vain jotain tiettyä tietomallia käyttäviä NoSQL-tietokantoja. Kohdassa 5.2 käydään läpi NoSQL-tietokantojen tietoruvallisuusominaisuuksiin liittyviä tutkimuksia ja kohdassa 5.3 käydään läpi NoSQL-tietokantojen suorituskykyyn liittyviä tutkimuksia. Tutkimuksia on etsitty erilaisten hakupalveluiden, kuten Googlen, Google Scholarin, IEEE Xploren ACM Digital Libraryn kautta. Näistä on tarkasteltu tämän tutkimuksen kannalta oleellisimpia ja uusimpia julkaisuja.

5.2 Tietoturvallisuusominaisuuksiin liittyvät tutkimukset

NoSQL-tietokantojen tietoturvaominaisuuksia ovat käsitelleet muun muassa Okman et al. [2011], Grolinger et al. [2013], Noiumkar ja Chomsiri [2014], Zahid et al. [2014] ja Wong [2014]. Kokonaisvaltaisin näistä tutkimuksista on Grolingerin et al. [2013] tekemä tutkimus, jossa on vertailtu merkittävimpiä NoSQL- ja NewSQL-tietokantoja. Tutkimuksessa tietokantojen merkittävyys on arvioitu DB-Ranking-sivuston sijoituksen mukaan ja IEEE:n julkaisusarjoissa esiintyvien viittausten mukaan. Mainittujen tutkimusten perusteella voidaan todeta, että NoSQL- ja NewSQL-tietokannoissa tietoturvaominaisuudet ovat yleensä ottaen varsin puutteellisia. Vain muutamat tutkituista tietokannoista sisältävät tarvittavat tietoturvaominaisuudet lainsäädännön asettamien vaatimusten täyttämiseksi.

Grolingerin et al. [2013] tutkimukseen valitut tietokannat ovat avain-arvo-varastoista Redis [2015], Memcached [2009], BerkeleyDB [2015], Voldemort [2015] ja Riak [2015]. Sarakeperhevarastoista mukana ovat Cassandra [2015], HBase [2015], DynamoDB [2015] ja SimpleDB [2015]. Dokumenttivarastoista MongoDB [2015], CouchDB [2015] ja Couchbase server [2015]. Graafitietokannoista mukana ovat Neo4J [2014], HyperGraphDB [2010], AllegroGraph [2015] ja NewSQL-tietokannoista VoltDB [2015], Spanner [Corbett el al., 2013], Clustrix [2015] ja NuoDB [2015]. Tutkimuksen mukaan sarakeperhevarasto Cassandran Enterprise-versio ja NewSQL-tietokanta NuoDB sisältävät kattavimmat tietoturvaominaisuudet. Nämä tietokannat tukevat tietokantatiedostojen kryptausta, yhteyden kryptausta niin asiakas/palvelin kuin palvelin/palvelin yhteyksissä, käyttäjien

tunnistamista ja valtuuttamista ja auditointia. Auditointiominaisuudet ovat lainsäädännön kannalta erityisen hyödyllisiä, sillä näitä ominaisuuksia tukeva tietokanta osaa pitää kirjaa tietokannan käytöstä. Cassandran ja NuoDB:n lisäksi mielenkiintoinen tietokanta on myös tutkimuksessa mainittu BerkeleyDB. Tietoturvaominaisuuksista se tukee vain datan kryptausta, mutta se on sulautettu tietokanta, joka voidaan integroida osaksi sitä käyttävää sovellusta ja näin voitaisiin rajata tietokantaan pääsy vain sovellukselle, johon kanta on integroitu. Tällä tavalla tietoturvaan liittyvät vaatimukset on ratkaistu nykyisessä järjestelmässä.

Muissa mainituista tutkimuksista on tutkittu paljon suppeampaa joukkoa tietokantoja. Okmanin et al., [2011] ja Wongin [2014] tutkimuksissa on vertailtu vain Cassandraa ja MongoDB:tä ja näissä on vertailtu samoja asioita kuin Grolinger et al. [2013] tutkimuksessa. Noiumkarin ja Chomsirin [2014] tutkimuksessa on tutkittu viiden www.bigdataanalyticsnews.com-sivuston mukaan suosituimman avoimen lähdekoodin tietokannan tietoturvaa. Nämä viisi tietokantaa tässä tutkimuksessa ovat MongoDB, Cassandra, CouchDB, Hypertable [2015] ja Redis. Näistä Hypertable ei ole mukana Grolingerin et al. [2013] tutkimuksessa. Hypertablessakaan ei tämän tutkimuksen mukaan ole datatiedostojen kryptausta, eikä asiakas/palvelin ja palvelin/palvelin yhteyksien kryptausta. Mielenkiintoisena tässä tutkimuksessa on myös, että tutkimuksessa on vertailtu haavoittuvuutta komentosarjoilla tehtäville injektioille (eng. script injection) ja palvelunestohykkäyksille (eng. denial of service attack). Näissä Hypertable ja Redis olivat vahvoilla. Näillä ominaisuuksilla ei verkkotapahtumien keräysjärjestelmässä tosin ole suurta merkitystä, sillä tietokannan sisältöön ei ole julkista pääsyä.

Zahidin et al. [2014] tutkimuksessa lähtökohtana on ollut tutkia suosituimpien hajautettujen (eng. sharded) tietokantojen tietoturvaa. Tutkitut tietokannat ovat MongoDB, Redis, CouchDB, Cassandra, HBase ja Couchbase server eli samat, jotka sisältyvät Grolingerin et al. [2013] tutkimukseen. Mitään merkittäviä asioita ei tässä tutkimuksissa tule esille muihin tutkimuksiin nähden. Vertailua ovat tehneet myös Kadebu ja Mapanga [2014] MongoDB:n, Cassandran, Neo4J:n ja HBasen tietoturvaominaisuuksista. Nämä tietokannat sisältyvät aikaisemmin mainittuun Grolingerin et al. [2013] tutkimukseen, mutta Kadebu ja Mapanga ovat antaneet myös ehdotuksia puutteiden korjaamiseksi.

5.3 Suorituskykyyn liittyvät tutkimukset

NoSQL-tietokantojen suorituskykyvystä löytyy varsin kirjava joukko erilaisia tutkimuksia. Mitään todella kattavaa tutkimusta, jossa vertailtaisiin kaikkia merkittävimpiä tietokantoja, ei löydy. Suorituskykyä vertailevissa tutkimuksissa on keskitytty vain muutamiin tietokantoihin. Paljon tutkittuja tietokantoja ovat Cassandra, HBase ja MongoDB. Viimeisimpiä ja laajimpia NoSQL-tietokantojen suorituskyvystä tehtyjä tutkimuksia ovat tehneet ainakin Datastax [2013], Nelubin ja Engber [2013], Li ja Manoharan [2013], Abramova et al. [2014] ja Klein et al. [2015].

Datastaxin [2013], tutkimuksessa on vertailtu Cassandran versiota 1.1.6, HBasen versiota 1.1.1 ja MongoDB:n versiota 2.2.2. Testityökaluna käytettiin Yahoon Cloud Serving Benchmarkia. Tietokannoille tehtiin muun muassa luku ja kirjoitustestejä eri kuormitusasteilla ja eri määrällä tietokantasolmuja. Cassandra pärjäsi näissä testeissä yleensä ottaen parhaiten. Erityisesti, kun solmujen määrää lisättiin, Cassandra sai paljon etumatkaa jättäen HBasen toiseksi ja MongoDB:n kolmanneksi. Nelubin ja Engberin [2013] tutkimuksessa on vertailtu Cassandraa, Couchbasea, MongoDB:tä ja Aerospikeä käyttäen Yahoon Cloud Serving Benchmarkin muunneltua versiota. Tässä tutkimuksessa Couchbase ja Aerospike päihittivät selvästi Cassandran ja MongoDB:n. Tutkimuksen testit suosivat kuitenkin avain-arvo-mallia, joten oli luonnollista, että avain-arvo-varastot Aerospike ja Couchbase suoriutuivat paremmin kuin sarakeperhevarsto Cassandra ja dokumenttivarasto MongoDB.

Li ja Manoharan [2013] ovat tutkineet MongoDB:n versiota 1.8.5, RavenDB:n [2015] versiota 960, CouchDB:n versiota 120, Cassandran versiota 1.1.2, Hypertablen versiota 0.9.6, Couchbasen versiota 1.8.0 ja MS SQL Expressin [2015] versiota 10.50.1600.1. Tutkimuksessa on vertailtu avain-arvo-parin luomiseen, lukemiseen, kirjoittamiseen ja poistamiseen kuluvaa aikaa eri tietokannoilla. Keskeisimpinä löytöinä tässä tutkimuksessa oli, että SQL-tietokanta MS SQL Express suoriutui testeissä joitakin NoSQL-tietokantoja paremmin. Kaikki NoSQL-tietokannat eivät siis tarjoa tämän tutkimuksen mukaan nopeusetuja SQL-tietokantoihin nähden vaikka NoSQL-tietokannat ovat paremmin optimoituja avain-arvo-pohjaisille operaatioille. Toisena löytönä oli, että tietokantojen tietomalli ei korreloi suorituskyvyn kanssa. Grolingerin et al. [2013] mukaan hyvät tietoturvaominaisuudet sisältävän Cassandran tulokset olivat keskitasoa. Parhaiten NoSQL-tietokannoista menestyivät MongoDB ja Couchbase luku-, kirjoitus- ja poistotesteissä. Huonoiten näissä pärjäsivät RavenDB ja CouchDB.

Abramova et al. [2014] ovat vertailleet Cassandran versiota 1.2.1, HBasen versiota 0.94.10, MongoDB:n versiota 2.4.6, OrientDB:n [2015] versiota 1.5 ja Rediksen versiota 2.6.14. Näistä OrientDB:tä [2015] voi käyttää dokumenttitietokantana ja graafitietokantana. Tietokantoja on testattu

Yahoo! Cloud Serving Benchmark -ohjelmistolla [Cooper et al., 2010]. Sovelluksen testeistä on käytetty luku- ja päivitysoperaatioita kuormittavia testejä ja näissä testeissä käytettiin testidatana 600 000 tietuetta. Kaiken kaikkiaan Redis oli selvästi nopein testatuista tietokannoista, Cassandra toiseksi nopein, kolmantena HBase, neljäntenä MongoDB ja kaikkein hitain oli OrientDB. Syynä tähän oli, että tietokanta vei enemmän resursseja kuin mitä testiympäristö antoi. Tutkimuksessa jaoteltiin tietokannat niihin, jotka olivat optimoitu lukuoperaatioille ja niihin, jotka olivat optimoitu päivitysoperaatioille. Ensimmäiseen joukkoon kuuluivat MongoDB, Redis ja OrientDB ja jälkimmäiseen joukkoon kuuluivat Cassandra ja HBase.

Klein et al., [2015] ovat vertailleet MongoDB:n versiota 2.2, Cassandran versiota 2.0 ja Riakin versiota 1.4. Tietokantoja testattiin 600 000 tietueella ja testisovelluksessa käytettiin Yahoo! Cloud Serving Benchmarkin ohjelmistokehystä. Testitulokset otettiin yhdeksän solmun kokoonpanoilla. Vahvan eheyden sisältävillä kokoonpanoilla tehdyissä testeissä Cassandran suorituskyky oli selvästi paras luku- ja kirjoitustesteissä, kun testisovelluksen säikeiden määrää lisättiin. Toisena tuli Riak ja kolmantena MongoDB. Toisaalta Cassandralla oli myös suurin viive luku ja kirjoitusoperaatioissa Riakin ollessa 5 kertaa nopeampi ja MongoDB:n ollessa 4 kertaa nopeampi. Syyksi Cassandran tehokkuuteen mainittiin ensimmäisenä sen tiivistepohjainen hajautus, josta oli testissä etua MongoDB:n hajautukseen nähden. Toisekseen Cassandran indeksointiominaisuudet mahdollistivat viimeksi kirjoitettujen tietueiden nopean haun ja kolmanneksi Cassandran vertaisverkkopohjainen arkkitehtuuri mahdollistivat tehokkaan koordinoinnin luku- ja kirjoitusoperaatioissa eri solmujen välillä.

5.4 Tietokantojen valinta ja rajaus

NoSQL-tietokantoja on markkinoilla varsin runsaat määrät. Esimerkiksi www.nosql-database.org-sivustolla on tätä työtä kirjoitettaessa listattuna 150 NoSQL-tietokantaa ja 18 NewSQL-tietokantaa. Tutkimuksessa käytettävät tietokannat pitää valita tästä joukosta. Koska nykyiset sovellukset hakevat tietokannasta dataa SQL-lauseita käyttäen, tietokannan olisi hyvä olla yhteensopiva SQL:n kanssa tai sisältää vastaavanlainen kyselyliittymä siten, että sen avulla voidaan toteuttaa kyselysovellusten vaatimat kyselyt.

Tietokantaan tuotava data sisältää paljon raportteja elementteihin tulevista puhelutiedoista. Tietokantoja tutkittaessa käytetään pohjana RNC-verkkoelementtien lähettämää RRC/RAB-raporttia. RNC on resurssien käyttöä ja eheyttä valvova radioliityntäverkon verkkoelementti. RRC/RAB-raportti sisältää puheluiden tunnistetiedot ja paljon teknistä tietoa puheluun liittyen. Tällainen raportti tehdään aina

puhelujen yhteydessä. Nykyisessä tietokannassa tästä raportista on käytössä kolme eri tyyppiä, jotka vastaavat kunkin RNC-elementin versioita. Uusimman version raportissa on 95 kenttää per tietue.

RRC/RAB-raportista halutaan hakea esimerkiksi vikatiedot jonkun tietyn IMEI-numeron sisältävälle laitteelle. Hyvä esimerkkikysely voisi olla vaikka ”hae raportit, jossa IMEI alkaa merkkijonolla x tai loppuu merkkijonoon y” tai ”hae kaikki raportit tietyssä IMEI-avaruudessa”. Tietokannan on pystyttävä tallentamaan tämän tyyppisiä raportteja niistä pitää pystyä hakemaan tietoa halutuilla tavoilla. Tietokannassa ei ole suhteita eri taulujen välillä, vaan jokainen taulu sisältää yhden itsenäisen raportin elementeiltä tulevasta datasta.

Jo näistä edellä esitetyistä ominaisuuksista seuraa, että yksinkertaista avain-arvo-varasto-mallia käyttävät tietokannat eivät ole kovin hyviä edellä mainitun tyyppiselle datalle. Jos tietokanta koostuu vain avaimista ja niiden alta löytyvistä arvoista, tiedon käsittely jouduttaisiin tekemään tietokannan ulkopuolella ohjelmakoodissa. Tällaisen datan kanssa käsittely menee varsin hankalaksi ja näin ollen avain-arvo-varastojen avulla tuskin saadaan kovinkaan suurta nopeusetua nykyisiin tietokantoihin nähden. Lisäksi järjestelmän SQL-pohjaisten sovellusten toimintaa jouduttaisiin muuttamaan nykyiseen nähden, jotta se saataisiin toimimaan avain-arvo-varaston kanssa. Joissakin toteutuksissa, kuten BerkeleyDB:ssä on mahdollisuus käyttää SQLite-kerrosta SQL-kyselyjen tekemiseen, mutta tässä tutkimuksessa avain-arvo-varastot jätetään kuitenkin vähemmälle huomiolle.

Dokumenttivarasto voisi sopia paremmin tämän tyyppiselle datalle. Koska dokumenttivarastossa dataa tallennetaan JSON-muotoisina rakenteina, koko raporteista voitaisiin muodostaa tietokantaan kokoelmia. Dokumenttivarastoista pystyy lisäksi tekemään SQL-tyyppisiä hakuja. Huonona puolena dokumenttivarastoissa on kuitenkin tilan käyttö. Jos otetaan 240 000 RRC/RAB-raporttia, vertaillaan niiden viemää tilaa nykyisessä tietokannassa ja JSON-formaatissa, huomataan, että nykyisessä tietokannassa tämä määrä raportteja vie noin 78 megatavua tilaa. JSON-formaatissa tilaa menee kuitenkin 517 megatavua. XML-formaatissa tilaa kuluu vielä enemmän, 585 megatavua. Kummassakin tapauksessa dokumenttirakenteiden määritykset vievät tilaa niissä olevan datan lisäksi.

Graafitietokanta ei myöskään ole hyvä valinta tässä tapauksessa, koska graafitietokannan tietomalli on suunniteltu paljon suhteita sisältävälle datalle. Kantaan tallennettavissa raporteissa ei ole ollenkaan suhteita, joten graafitietokantojen ominaisuuksista tuskin olisi suurta hyötyä tässä tapauksessa. Tietokanta tulisi koostumaan solmuista, joiden välillä ei ole kaaria ollenkaan. Graafitietokantoja myös kehitetään siten, että niissä keskitytään erityisesti relaatiotyyppisen datan käyttämiseen, joten niitä

käyttämällä tuskin saadaan tehokkuushyötyä tässä tapauksessa. Pidemmällä aikavälillä jonkin graafitietokannan käytöstä voisi tulla hankaluuksia, sillä graafitietokantojen kehityksessä todennäköisesti painotutaan relaatioita sisältävän datan vaatimuksiin.

Sarakeperhevarastot sopivat neljästä NoSQL-tietomallista parhaiten järjestelmän käyttämälle datalle, koska järjestelmässä käsiteltävä raportti voidaan helposti sisällyttää sarakeperheen riviksi. Sarakeperhevarastojen ominaisuudet voisivat tuoda esimerkiksi nopeusetuja kyselyihin, koska yksi raportti voitaisiin hakea kysymällä oikeaa riviavainta. Lisäksi sarakeperhevarastoissa kuten Cassandrassa ja Hypertablessa käytetään SQL:n kaltaisia kyselykieliä CQL ja HQL. Näiden kanssa voitaisiin saada nykyinen järjestelmä toimimaan ilman suuria muutoksia lähdekoodiin.

Uusissa tietokantatuotteissa myös NewSQL-tietokannat ovat järjestelmän kannalta mielenkiintoisia. Koska NewSQL-tietokannat tukevat perinteistä SQL-kyselykieltä ja muita relaatiotietokannan ominaisuuksia, siirtyminen NewSQL-tietokantaan tämän tyyppisessä järjestelmässä onnistuisi varsin vaivattomasti. Nykyinen relaatiotietokantaa käyttävä käyttöliittymä toimisi NewSQL-kannan kanssa ongelmitta. Lisäksi saataisiin kuitenkin NoSQL:n tuomat hajautusominaisuudet, jotka NewSQL tietokannat lisäävät perinteiseen SQL-tietokantaan. Hajautuksesta voisi olla verkkotapahtumien keräysjärjestelmässä hyötyä, sillä sama tietokanta voitaisiin levittää usean tiedonkerääjäpalvelimen yli.

5.5 Tutkittavat tietokannat

NoSQL-tietokannoista kohdan 5.1 mukaan Cassandrassa vaikutti olevan varsin hyvät tietoturvaominaisuudet, mikäli käyttää Cassandran Enterprise-versiota. Kohdan 5.2 mukaan Cassandra vaikutti myös useiden testien perustella olevan varsin nopea tietokanta muihin testeissä testattuihin tietokantoihin nähden. Lisäksi Cassandraa käytetään myös yrityksen eräässä toisessa projektissa, joten yrityksestä löytyy Cassandraan liittyvää osaamista. Näin ollen NoSQL-tietokannoista valittiin tutkittavaksi Cassandra. NewSQL-tietokannoista ei löytynyt yhtä paljon tietoa kuin NoSQL-tietokannoista. Kohdassa 5.1 mainitussa Grolingerin et al. [2013] tutkimuksessa oli tarkasteltu muutamia NewSQL-tietokantoja. Niistä NuoDB-nimisessä kannassa oli tutkimuksen mukaan parhaat tietoturvaominaisuudet. Toisaalta NuoDB:ssä datan hajautus ei ole saman tutkimuksen mukaan käyttäjän hallittavissa, joten siinä mielessä tietokanta ei välttämättä sovellu osaksi verkkotapahtumien keräysjärjestelmän arkkitehtuuria. Yrityksessä kiinnostuttiin Parstream-nimisestä tietokannasta, joka voitti yrityksen innovaatiokilpailun. Parstreamin web-sivujen [2015] mukaan tietokannan ominaisuudet mahdollistavat maantieteellisesti hajautetun tietokantaratkaisun toteuttamisen. Web-sivun mukaan

tietokannan avulla voitaisiin rakentaa verkkotapahtumien keräysjärjestelmän kaltainen ratkaisu, jossa tietokanta hajautetaan lähelle datalähdettä.

5.6 Cassandra

Cassandran tietomalli koostuu avainavaruuksista, joihin luodaan sarakeperheitä. Avainavaruudet vastaavat pääpiirteittäin perinteisten tietokantojen kaavioita ja sarakeperheet tauluja. Erityisesti Cassandran perinteistä kyselykäyttöliittymää käytettäessä sarakeperhe ajatellaan hieman erilailla kuin perinteinen tietokantataulu. Sarakeperheet koostuvat avain-arvo-pareista, joita kutsutaan sarakkeiksi. Nämä avain-arvo-parit ryhmitellään riveiksi ja ryhmät tunnistetaan riviavaimella. Cassandran sarakeperheet voivat olla staattisia tai dynaamisia. Staattisissa sarakeperheissä jokaisella rivillä on aina samassa järjestyksessä tietyn nimiset sarakkeet. Näin se muistuttaa perinteistä relaatiotietokannan taulua. Dynaamisissa sarakeperheissä jokaisella rivillä voi olla erinimisiä sarakkeita ja niitä voi lisätä ja poistaa mielivaltaisesti.

Taulukossa 1 on kuvattu staattinen sarakeperhe ja taulukossa 2 dynaaminen sarakeperhe. Punaisella pohjalla näkyvät riviavaimet, vihreällä pohjalla sarakeavaimet ja valkoisella pohjalla sarakeavainten takana olevat arvot. Kuten huomataan, staattisessa sarakeperheessä sarakeavaimet ovat jokaisella rivillä samassa järjestyksessä, mutta dynaamisessa sarakeperheessä järjestys voi vaihdella riveittäin. Cassandran vanhassa kyselykäyttöliittymässä taulun 1 ensimmäinen rivi saadaan haettua kokonaisuudessaan komennolla get[’1’]. Sarakkeen a arvo riviavaimen 1 tunnistamassa sarakeperheessä voitaisiin asettaa arvoksi 10 komennolla set[’1’][’a’]=10 ja haettua komennolla get[’1’][’a’]. Koko sarakejoukon saa siis Cassandrassa haettua varsin tehokkaasti, jos tietää sarakejoukon tunnistavan riviavaimen.

Taulukko 1: Staattinen sarakeperhe.

1	a	b	c	d
	10	20	30	40
2	a	b	c	
	10	20	30	
3	a	b	c	d
	10	20	30	40

Taulukko 2: Dynaaminen sarakeperhe.

1	a	b	c	d
	10	20	30	40
2	e	f		
	10	20		
3	d	c	b	a
	10	20	30	30

Cassandrassa sarakkeiden nimillä on väliä sarakeperheen rakenteen kannalta. Cassandra järjestää sarakkeet niiden nimien mukaan automaattisesti. Jos taulukon 2 rivin 3 tunnistamaa sarakeperhettä kysellään sen riviavaimella, Cassandra antaa sarakkeet automaattisesti niiden nimen mukaan järjestettynä. Eli tässä tapauksessa riviavainta 3 kyseltäessä palautettaisiin sarakkeet kuitenkin aakkosjärjestyksessä a,b,c,d. Hakutuloksissa sarakkeiden järjestystä ei voi siis itse määrätä, kuten relaatiotietokannoissa ja tämä pitää ottaa huomioon dataa käsiteltäessä.

Versiosta 0.8 alkaen, tietoa on voitu käsitellä Cassandrassa SQL-kieltä muistuttavan CQL-kyselykielen avulla [Evans, 2008]. Tämän jälkeen Cassandraa on voinut käyttää perinteisen SQL-tietokannan tapaan. CQL-kielellä tauluja käsitellään SQL-kieltä muistuttavien lauseiden avulla. Sarakeperheen luonti, poisto ja päivitys onnistuvat SQL:stä tutuilla CREATE-, UPDATE- ja DELETE-lauseilla ja tietoa voi kysellä SELECT-lauseilla. JOIN-lauseita ei tosin edelleenkään tueta, sillä sarakeperheitä ei voi edelleenkään yhdistellä, koska vierasavaimia ei ole. Vaikkakin CQL-kielen myötä Cassandrassa on siirrytty enemmän tavallisen relaatiotietokannan mallia kohti, Cassandra on edelleen tekniseltä toiminnallisuudeltaan sarakeperhevarasto.

Staattiset sarakeperheet voidaan toteuttaa luomalla sarakeperhe ja määrittämällä sille pääavain. Tällöin sarakeperhe muistuttaa ulkoisesti SQL-tietokannan tietokantataulua, mutta Cassandra käsittelee sitä perinteisen sarakeperheen tapaan siten, että pääavainta käytetään riviavaimena. Dynaaminen sarakeperhe voidaan toteuttaa CQL-kielen avulla siten, että määritellään yhdistelmäavain (engl. compound key) kahdesta sarakkeesta. Käytännössä tämä tapahtuu määrittämällä kaksi saraketta osaksi pääavainta. Tällöin toinen näistä avaimista toimii riviavaimena ja toinen sarakeavaimena. Esimerkiksi, jos halutaan tallentaa elokuvien saamia arvosanoja ja arvosteluja on tehty tiettyinä ajankohtina, voidaan elokuvan nimestä ja arvostelun aikaleimasta tehdä yhdistelmäavain. Tällöin Cassandra käsittelee

elokuvan nimeä riviavaimena ja arvostelun aikaleimaa sarakeavaimena. Sarakeavaimen takaa löytyy elokuvan saama arvosana. [Klishin and Monadovič, 2015]

5.7 Parstream

Parstream on varsin uusi tietokanta. Siitä ei löydy tätä tutkimusta tehdessä juurikaan mainintoja, jos etsii tietokantaan liittyviä tutkimusartikkeleja. Parstream muistuttaa perinteistä tietokantaa siten, että Parstream on täysin SQL:2008 yhteensopiva. Tietokannan avulla voidaan toteuttaa perinteinen relaatiotietokanta, sillä yhteensopivuus SQL-standardin kanssa mahdollistaa tämän. Perinteisen relaatiotietokannan ominaisuuksien lisäksi tietokanta sisältää kuitenkin suorituskykyä parantavia ominaisuuksia kuten taulujen osioinnin, bitmap-indeksoinnin ja oman ratkaisun hajautetun tietokannan toteuttamiseksi. Tietokantatyypiltään Parstream on periaattessa Aslettin [2011] määrittelemä NewSQL-tietokanta, sillä se pyrkii yhdistämään hajautusominaisuudet perinteiseen SQL-tietokantamalliin. ACID-ominaisuuksia Parstream ei kuitenkaan tue, mutta tässä tutkimuksessa Parstream luokitellaan NewSQL-tietokannaksi, sillä se soveltuu ominaisuuksiensa puolesta parhaiten tähän kategoriaan.

Parstream jakaa tietokannan loogisiin ja fyysisiin lohkoihin. Sama tietokantataulu jaetaan moneen eri loogiseen osioon, jotka yhdistettynä muodostavat koko tietokantataulun. Loogiset osiot koostuvat fyysisistä osioista. Fyysisiä osioita käytetään datan hallitsemiseksi tietokantatauluissa. Jokainen datan tuonti luo uuden fyysisen osion tietokantatauluun. Tietokantataulun osiointi määritetään tietokantataulua luotaessa PARTITIONED BY -lauseella, jossa annetaan sarakkeet, joiden mukaan osiointi tehdään. Sopivalla osioinnilla voidaan nopeuttaa tietokantakyselyitä huomattavasti. Otetaan esimerkiksi jonkin yhdistyksen jäsentietokanta. Jäsentietokannassa on sarake, jossa on tieto, onko jäsenmaksu maksettu ja sen arvo voi olla kyllä tai ei. Nyt, jos yhdistyksen tietokantaan tehdään paljon kyselyitä liittyen jäsenmaksun maksamiseen, kannattaa tietokantataulu osioida jäsenmaksutietoa koskevan sarakkeen mukaan. Tällöin taulu osioidaan kahteen osaan. Toisessa on jäsenet, joiden jäsenmaksu on maksettu ja toisessa jäsenet, joiden jäsenmaksua ei ole maksettu. Nyt, kun halutaan saada tiedot niistä jäsenistä, joiden jäsenmaksu on maksettu, tietokannan ei tarvitse käsitellä kyselyssä maksamattomien jäsenten osiota. Osion pudottaminen pois kyselystä saattaa nopeuttaa kyselyä huomattavasti. [Parstream, 2014]

Parstreamin yksi tärkeistä suorituskykyä parantavista ominaisuuksista on tuki bittikarttaindeksoinnille. Parstreamissa on tätä ominaisuutta varten toteutettu oma patentoitu ratkaisu [Bienert et al., 2013]. Bittikarttaindeksoinnin ideana on, että sarakkeen arvoista luodaan ykkösiä ja nollia sisältävä bittikartta. Bittikarttaindeksiä havainnollistetaan taulukossa 3.

Taulukko 3: Jäsenmaksutiedoista tehty bittikartta.

Jäsennumero	Maksettu	Bittikartta	
		Kyllä	Ei
1	Kyllä	1	0
2	Kyllä	1	0
3	Ei	0	1
4	Ei	0	1
5	Kyllä	1	0
6	Kyllä	1	0

Taulukossa 3 nähdään maksettu-sarakkeesta muodostettu bittikarttaindeksi. Siinä on luotu kaksi bittikarttaa vaihtoehdoille kyllä ja ei. Kyllä-vaihtoehdon bittikartassa merkitään yksi niiden jäsenten kohdalle, joiden jäsenmaksu on maksettu ja nolla niiden kohdalle, joiden jäsenmaksu ei ole maksettu. Samalla tavalla tehdään Ei-vaihtoehdon kanssa. Nyt kun tietokannasta haetaan sellaisten käyttäjien tiedot, jotka ovat maksaneet jäsenmaksun, voidaan käydä läpi vain kyllä-vaihtoehdon bittikartta ja ottaa sieltä vain ne rivit joilla on kyllä-bittikartassa arvo yksi. Bittikartan avulla voidaan myös säästää tilaa, sillä se voidaan pakata käyttämällä jakson pituuden koodausta (engl. Run-length Encoding) [Dipperstein, 2014].

Turvallisuusominaisuuksissa Parstreamissa on puutteita. Tietokanta ei tue datatiedostojen salausta. Merkkijonotyyppisten sarakkeiden sisällön saa datatiedostoista näkyviin selvästi ymmärrettävässä muodossa tekstieditorilla. Tietokantayhteyksien salausta ei myöskään tueta. Käyttäjien autentikointi ja käytön kirjaaminen lokiin on kuitenkin tuettu.

5.8 Datan hajauttaminen Cassandrassa ja Parstreamissa

Cassandralla hajautettu järjestelmä voidaan toteuttaa asentamalla Cassandra kaikille tietokantapalvelimille ja valitsemalla jokin näistä palvelimista siemensolmuksi (engl. seed node). Muut solmut käyttävät siemensolmuja tutkiakseen tietokannan rakenteen ja löytääkseen toisensa. Datan hajautus tapahtuu Cassandrassa sarakeperheen pääavaimen mukaan. Se, miten hajautus tapahtuu, riippuu osioijasta (engl. partitioner). Cassandrassa voi valita kolmesta osioijasta: RandomPartitioner, Murmur3Partitioner ja ByteOrderedPartitioner. RandomPartitioneria käytettäessä Cassandra laskee pääavaimen arvoista MD5-tiivisteen ja Murmur3Partitioneria käytettäessä Murmur3_128-tiivisteen. Solmut jaetaan eri arvoalueisiin laskettujen tiivisteiden mukaan ja aina, kun uutta dataa tulee sisään

tietokantaan, se allokoidaan siihen solmuun, jonka arvoalueeseen se kuuluu. ByteOrderedPartitioner taas järjestää pääavaimet järjestykseen sen mukaan, mikä pääavaimen arvo on bitteinä. [Datastax, 2015a]

Jos Parstreamilla tehdään klusteroitu järjestelmä, hajautus tapahtuu sen mukaan, mikä tai mitkä sarakkeet määritetään hajautuksen perusteeksi. Tietokanta tukee staattista hajautusta ja dynaamista hajautusta. Staattisessa hajautuksessa määritellään etukäteen, mikä arvo menee millekin solmulle. Esimerkiksi, jos taulussa on sarake ”numero” ja siihen tulee arvoja väliltä 1–5, voidaan määritellä, että 1 viedään solmulle 1, 2 viedään solmulle 2 ja niin edelleen. Tämän lisäksi voidaan määritellä vaihtoehtoiset solmut, joille tietoa hajautetaan. Esimerkiksi arvo yksi hajautettaisiin solmuille 1, 2 ja 3 siten, että jos solmu 1 ei vastaa, niin arvo viedään solmulle 2 ja jos sekään ei vastaa, arvo viedään solmulle 3. Staattista hajautusta määriteltäessä määritellään usein ensisijainen solmu, jolle arvo viedään ensisijaisesti ja vaihtoehtoiset solmut käymällä läpi kaikki solmujen permutaatiot. Dynaamisessa hajautuksessa ei tarvitse määritellä välttämättä mitään hajautusasetuksia, vaan tietokanta osaa hoitaa arvojen hajautuksen automaattisesti. Dynaamisessa hajautuksessa voidaan määritellä käytettävien varasolmujen määrä. Oletuksena käytetään yhtä varasolmua, mikäli ensisijainen solmu ei ole saatavilla.

Parstreamissa hajautettu tietokanta voidaan toteuttaa kyselysolmujen (engl. query node) ja tuojasolmujen avulla. Kyselysolmut vastaavat nimensä mukaisesti tietokantakyselyihin ja import-solmut vastaavat tiedon viennistä tietokantaan. Parstreamissa solmun asetukset määritellään solmun kotihakemistosta löytyvästä ini-tiedostosta. Tiedostossa määritellään jokaiselle solmulle rank-arvo ja siitä solmusta, jolla on pienin arvo, äänestetään johtaja. Johtaja on vastuussa järjestelmän tilan tutkimisesta ja viestimisestä muille solmuille, lohkojen yhdistämisestä ja epäonnistuneiden datan tuontien ja yhdistämisten hallinnoinnista. Jos johtajasolmuun tulee vika, uusi johtajasolmu äänestetään seuraavaksi siitä solmusta, jolla on suurin rank-arvo. [Parstream, 2014]

5.9 Verkkotapahtumien keräysjärjestelmän tietokannan toteuttaminen Cassandran ja Parstreamin avulla

Nykyisellään verkkotapahtumien keräysjärjestelmässä on yksi tietokanta jokaista tiedonkerääjäpalvelinta kohti. Datan kyseleminen eri palvelimelta on ratkaistu järjestelmän omien sovellusten avulla. Uusien NoSQL- ja NewSQL-tietokantasovellusten avulla voitaisiin kuitenkin toteuttaa tiedonkeräyspalvelin hajautettuna ratkaisuna eli tietokanta sijaitsisikin yhden fyysisen palvelimen sijasta usealla palvelinsolmulla. Tästä ratkaisusta voisi olla hyötyä, sillä kyselyt voitaisiin

tehdä vain yhdelle palvelimelle, jossa on tietokantasolmu ja näin ollen tietokantakyselyssä tiedon hakua usealta palvelimelta ei tarvitsisi toteuttaa sovelluksessa, vaan tietokanta hoitaisi tämän.

Kuten luvussa 4 esiteltiin, verkkotapahtumien keräysjärjestelmässä on välittäjäkerros ja tietovarastokerros. Koska verkkotapahtumien keräysjärjestelmässä kyselyt tehdään välittäjäkerroksen kautta, hajautettu tietokanta pitäisi pystyä toteuttamaan siten, että välittäjäkerroksella sijaitsisi vain kyselyihin käytettävä tietokantasolmu ja tietovarastokerroksella sijaitsisivat solmut, joihin tuodaan ja joissa säilytetään dataa. Kyselyihin käytettävä tietokantasolmu toimisi kyselyiden välittäjänä hajautetussa tietokannassa ja myös koordinaattorina alemman kerroksen solmujen välillä. Sekä Cassandran, että Parstreamin avulla voitaisiin toteuttaa nämä vaatimukset täyttävä järjestelmä.

Tässä tutkimuksessa oletetaan, että jokaisella tietokantaan dataa sisään tuovalla sovelluksella on oma tunnisteensa. Tunnisteelle tehdään tietue jokaiseen sisään tuotavaan raporttiin. Näin tunnistetta voitaisiin käyttää hajautusavaimena, jonka perusteella tietokanta päättää, mihin solmuun data viedään. Tunnisteiden avulla voitaisiin hallita tietokannan datan hajautusta, mikä olisi verkkotapahtumien keräysjärjestelmän kannalta tärkeää. Järjestelmässä halutaan, että data sijaitsee fyysisesti määrätyssä paikassa.

Cassandrassa verkkotapahtumien keräysjärjestelmän hajautettu tietokanta voitaisiin toteuttaa kohdan 5.8 ensimmäisessä kappaleessa mainitun ByteOrderedPartitionerin avulla. ByteOrderedPartitionerilla voitaisiin määritellä jokaiselle solmulle tietyt arvoalueet ja näin voitaisiin määrittää miten data hajautetaan. Tunnisteesta tehtäisiin sen sarakeperhevaraston pääavain, mihin raporttidataa viedään. Otetaan esimerkiksi tilanne, jossa käytössä olisi yksi solmu p välittäjäkerroksella ja tietovarastokerroksella solmut a, b ja c. Tiedonkeräyspalvelimeen a veisi dataa sovellukset, joiden tunnisteet ovat 1 ja 2, b:hen 11 ja 12 ja c:hen 21 ja 22. Nyt jos tällainen järjestelmä toteutettaisiin Cassandraa ja ByteOrderedPartitioneria hyväksi käyttäen, asennettaisiin Cassandra kaikille solmuille a b, c ja p. Cassandran asetuksista valittaisiin osiosijaksi ByteOrderedPartitioner. Jokaiselle solmulle määritettäisiin arvoalueet siten, että arvot a:n arvoalue olisi 1–10, b:n arvoalue 10–20 ja c:n arvoalue 20–30. Solmun p arvoalue asetettaisiin sellaiseksi, että sinne ei mene mitään dataa, sillä solmua halutaan käyttää vain kyselysolmuna. Näin tunnisteita 1 ja 2 käyttävät sovellukset veisivät dataa solmulle a, tunnisteita 11 ja 12 käyttävät sovellukset solmulle b ja tunnisteita 21 ja 22 solmulle c.

Parstreamissa vastaavan järjestelmän toteuttaminen onnistuisi helposti kohdan 5.8 toisessa kappaleessa mainitun staattisen hajautuksen avulla. Oletetaan edellisessä kappaleessa mainittu tilanne. Parstream

asennettaisiin kaikille solmuille a, b, c ja p. Tietokantatauluun, joihin dataa tuodaan, määritettäisiin hajautusasetukset siten, että arvot 1 ja 2 viedään solmulle a, arvot 11 ja 12 solmulle b ja arvot 21 ja 22 solmulle c ja solmulle p ei määritetä vietäväksi mitään arvoja. Parstreamin tapauksessa siis pitää määrittää mitkä arvot viedään millekin solmulle, kun taas Cassandrassa määritetään arvoalueet, joille arvot hajautetaan.

6 Tietokantojen testaus

6.1 Tutkimuksessa käytetty tietokantataulu ja sarakeperhevarasto

Testausta varten kumpaankin tietokantaan tehtiin rakenne, johon tietoa tallennetaan. Kuten kohdassa 5.6 mainittiin, Cassandrassa tästä rakenteesta käytetään nimeä sarakeperhe. Parstreamissa taas puhutaan tietokantatauluista perinteisten relaatiotietokantojen tapaan. Cassandraa varten luodut sarakeperhevarastot ja Parstreamia varten luodut tietokantataulut eivät täysin vastaa toisiaan tietotyyppien osalta, eivätkä ne vastaa myöskään nykyisessä tietokannassa käytettyä tietokantataulua, tietokannoissa on erilainen tuki eri tietotyypeille. Testeissä käytetyssä sarakeperhevarastossa ja tietokantataulussa käytettiin nimeä RRCRABTEST.

Cassandrassa sarakeperhevarastoja luotaessa täytyy ottaa huomioon niissä käytetty pääavain. Tämä vaikuttaa siihen, miten Cassandra tallentaa datan. Tietotyypeistä Cassandra ei tue samoja kokonaislukutyyppejä kuin nykyisin käytössä oleva tietokanta. Nykyisin numeromuotoisen datan tallentamiseen käytetään osassa sarakkeita alle 32-bittisiä kokonaislukutyyppejä kuten esimerkiksi smallint, mutta Cassandrassa nämä kaikki pitää tallentaa 32-bittisenä int-tyyppinä, sillä pienempää tyyppiä ei ole käytettävissä. Merkkijonoille Cassandra tarjoaa text- ja varchar-tyypit. Kummankaan kokoa ei voi etukäteen määritellä, joten aikaisemman tietokannan varchar-arvojen kokomäärityksiä ei voida käyttää. [Datastax, 2015b]

Parstreamiin tauluja luotaessa täytyy myös suunnitella ensin se, minkä sarakkeiden suhteen tietokantataulu hajautetaan, sillä Parstream tallentaa tietokantataulun datan sen mukaan, mitä tietokantataulun sarakkeita on listattu CREATE TABLE -lauseen yhteydessä määriteltävässä PARTITIONED BY -lauseessa. Tietotyypeistä Parstream tukee paljon laajempaa joukkoa kuin Cassandra ja kokonaislukutyypeistä voidaan käyttää myös 8- ja 16-bittisiä int-tietotyyppejä. Parstreamissa sama RRC/RAB-raportti voidaan tallentaa pienempikokoisia tietotyyppejä käyttäen.

Kummassakin tietokannassa hajautusavaimena toimi seuraavassa kohdassa esiteltävän tietoa syöttävän sovelluksen tunniste, jolle tehtiin oma inserter_id-niminen sarake RRCRABTEST-sarakeperhevarastossa ja tietokantataulussa. Inserter_id:stä tehtiin ensimmäinen osa Cassandran sarakeperhevaraston PRIMARY KEY -määrityksessä ja Parstreamin taulun PARTITIONED BY -määrityksessä. Cassandrassa inserter_id toimi siis riviavaimena. Toisena sarakkeena näissä määrityksissä käytettiin raportin aikaleimaa. Raportin aikaleiman käyttäminen tässä tapauksessa olisi

verkkotapahtumien keräysjärjestelmän kannalta hyvä valinta, sillä verkkotapahtumien keräysjärjestelmästä haetaan usein raportteja tietyltä aikajaksolta. Näin aikaleiman käyttäminen toisi nopeutta aikaväleiltä tehtäviin tietokantahakuihin.

Aikaleiman sisältävän sarakkeen nimi on report_time. Cassandrassa kaksiosainen pääavain eli yhdistelmäavain muodostaa kohdassa 5.6 kuvatun dynaamisen sarakeperheen. Tällöin Cassandra tallentaa datan siten, että inserter_id toimii riviavaimena ja report_time-sarakkeen aikaleima yhdistettynä jokaisen muun sarakkeen nimeen toimivat sarakeavaimina, joiden alta tietokantaan tallennetut arvot löytyvät. Näin ollen siis kaikkien arvojen, mitä RRCRABTEST-sarakeperhevarastosta löytyy, avaimina toimii yhdistelmä report_time:sarakkeen nimi. Näin toteutettuna esimerkiksi rrc_ipv4 sarakkeeseen tallennettu IPv4-osoite haetaan kannasta avainyhistelmällä inserter_id:n arvo, report_time:n aikaleima ja rrc_ipv4.

Cassandraan dataa vietäessä huomattiin mielenkiintoinen piirre Cassandran toiminnasta, kun dataa syötettiin pääavainmäärityksellä PRIMARY KEY(inserter_id, report_time). Cassandra ei koskaan kirjoita kantaan kahta eri riviä, joilla on sama pääavainyhdistelmä, vaan Cassandra kirjoittaa uudemman rivin vanhan päälle. Jos siis viedään useampi RRC/RAB-raportti tietokantaan siten, että kaikki raportit sisältävät samat inserter_id- ja report_time-arvot, kantaan jää näistä raporteista viimeisimmäksi viety. Tästä syystä Cassandran pääavainmääritykseen piti lisätä vielä kolmaskin sarake. Kolmas sarake vaihteli eri testien mukaan. Useissa testeissä käytettiin juoksevaa numeroa, joka tallennettiin sarakkeeseen rowindex ja toisissa käytettiin IMSI-arvoja, jotka tallennettiin sarakkeeseen rrc_imsi.

6.2 Tietokantoihin dataa syöttävä sovellus InserterDriver

Tietokantojen datan syöttönopeuden tutkimiseksi toteutettiin sovellus, joka generoi RRC/RAB-raportin sisältöä muistuttavaa dataa. Sovellus toteutettiin Java-kielellä ja siihen toteutettiin toiminnallisuudet sekä Cassandran, että Parstreamin testaamiseksi. Sovelluksessa voidaan valita kuinka monta riviä dataa generoidaan ja kuinka monta rinnakkaista säiettä käynnistetään datan syöttämiseksi. Lisäksi sovelluksessa voi valita haluaako dataa syöttää tietyn ajan verran vai tietyn rivimäärän verran. Sovelluksen nimeksi annettiin InserterDriver.

Datan syöttämiseksi Cassandraan käytettiin Datastaxin tarjoamaa Java-rajapintaa [Datastax, 2015c]. Cassandran datan vienti on toteutettu PreparedStatement- ja BoundStatement-lauseiden avulla. Dataa sisään vievä INSERT-lause muotoillaan PreparedStatement-lauseessa ja siihen sidotaan arvoja

BoundStatement-lauseessa. Erottamalla lauseen muotoilu ja arvojen sitominen saadaan hieman nopeusetuja, sillä INSERT-lause tarvitsee kääntää vain kerran. Dataa syötettäessä Cassandraan voidaan käyttää execute()- tai execute_async()-metodeja. Execute()-metodi on synkroninen ja jää odottamaan tietokannan vastausta. Execute_async() taas palauttaa ResponseFuture-objektin, johon vastaus tulee myöhemmin. Testeissä käytettiin execute_async()-metodia, sillä sen ajateltiin olevan nopeampi kuin execute().

Datan syöttämiseksi Parstreamiin käytettiin Parstreamin omaa Streaming Import -rajapintaa. Streaming Importia käytettäessä ei tarvitse muotoilla INSERT-lausetta, vaan yksinkertaisesti yhteys Parstreamiin avataan ja tietokannan rivi syötetään sisään Parstreamiin rawInsert()-metodin avulla listamuodossa. Jotta syötetyt rivit kirjoitettaisiin tietokantaan, pitää Parstreamissa kutsua commit()-metodia perinteisen relaatiotietokannan tapaan. Syötettävä data generoidaan metodissa listarakenteeseen. Numeroarvojen luomiseen käytetään Javan Random-luokkaa, jonka avulla generoidaan satunnaisia lukuja sarakkeisiin eri arvoalueilta. Arvoalueet perustuvat oikean RRC/RAB-raportin arvoalueisiin pääpiirteittäin. Parstreamia varten on täytynyt rajoittaa arvoalueita, sillä Parstream tulkitsee kokonaislukujen maksimiarvot NULL-arvoiksi. Esimerkiksi 16-bittisen kokonaisluvun maksimiarvoa 65535 ei voi viedä Parstreamiin, sillä Parstream tulkitsee tämän arvoksi NULL ja tulee virhe, joten tämän tyyppiset luvut generodaan arvojen 0–65534 väliltä.

Kokonaislukuarvojen lisäksi kantaan viedään IMSI-, TMSI-, IMEI-, IPv4-, IPv6-, URL- ja MSISDN-arvot. Näistä muut paitsi IMSI- ja URL-arvot generoidaan edellisessä kappaleessa mainittua Random-luokkaa käyttäen siten, että niiden muoto vastaa kuitenkin oikeaa muotoa. Esimerkiksi IMEI on tässä 15 numeroa pitkä satunnainen sarja. IPv4-arvoiksi generoidaan neljä satunnaista lukua arvojen 0–255 väliltä, joiden väliin laitetaan pisteet. IPv6-arvoiksi generoidaan 8 satunnaista neljän heksadesimaaliluvun sarjaa siten, että jokainen merkki voi saada arvon 0–F väliltä ja merkkijonot erotetaan kaksoispisteellä.

Dataa syöttävää sovellusta pitää kutsua komentoriviltä siten, että sovellukselle annetaan tietokannan nimi, säikeiden määrä, generoitavien rivien määrä, syötettävien rivien määrä per säie ja sleep-funktion aika sekunteina. Tietokannan nimi voi olla joko Cassandra tai Parstream. Sovellus koostuu luokasta InserterDriver, jossa on päämetodi ja luokasta Inserter, joka perii Javan Thread-luokan ja sitä käytetään datan sisään viemiseen siten, että useita Inserter-instansseja voidaan ajaa rinnakkain.

Kun InserterDriver käynnistyy, se lukee IMSI-listan ja URL-listan tekstitiedostoista listarakenteisiin. Testeissä on käytetty 10 miljoonaa IMSI-arvoa ja 9001 URL-arvoa. Tämän jälkeen sovellus käynnistää niin monta Inserter-säiettä kuin käyttäjä on antanut ja tietokannan nimi, generoitavien rivien määrä, syötettävien rivien raja ja luetut IMSI- ja URL-listat annetaan säikeille syötteenä. Inserter-säikeen tunnisteeksi annetaan juokseva numero. Tämä numero asetetaan tietokannassa inserter_id-sarakkeen arvoksi. Kun säikeet on luotu, sovellus antaa kaikille käskyn alkaa generoida dataa kutsumalla metodia startGeneration(). Datan generointi tehdään metodissa generateValues(), joka tuottaa listarakenteen, jossa on annetun rivimäärän verran generoituja arvoja. Tämän jälkeen InserterDriver odottaa, kunnes kaikki säikeet ovat saaneet datan generoitua ja generoimiseen kuluneen ajan. Sitten kaikille säikeille annetaan lupa viedä dataa sisään kutsumalla jokaiselle säikeelle metodia startInsertion().

Inserter-säikeessä datan sisään vienti tapahtuu iteroimalla kahta sisäkkäistä silmukkaa. Sisempi silmukka iteroi läpi listarakennetta, johon data on generoitu. Silmukkaa iteroidaan niin kauan kunnes kaikki listarakenteeseen generoidut rivit on viety sisään. Jokainen sisään viety rivi kasvattaa inserterCount-nimisen muuttujan arvoa. Ulompi silmukka tarkastaa onko rivejä viety sisään jo annetun rajan verran ja jos ei ole, mennään uudestaan sisäsilmukkaan, jossa aletaan viedä uudestaan dataa samasta listarakenteesta. Aikaleima, IMSI-arvo ja URL-arvo liitetään tässä vaiheessa listarakenteen arvoihin. Aikaleima asetetaan siten, että 15 riviä viedään samalla aikaleimalla, jonka jälkeen aikaleima päivitetään. IMSI- ja URL-arvot haetaan satunnaisesti niistä listarakenteista, joihin ne haettiin kun ohjelma käynnistettiin. Tietokantaan viedään siis sama rivijoukko useamman kerran paitsi, että aikaleima vaihtuu joka 15 rivin jälkeen ja IMSI ja URL vaihtelevat satunnaisesti. Tämä ratkaisu tehtiin siitä syystä, että jos halutaan viedä sisään esimerkiksi 1 000 000 000 riviä sisään, listarakenteen muodostaminen kestäisi kauemmin, jos kaikki rivit generoitaisiin erikseen.

Rivien sisään viennin aikana InserterDriverissä lasketaan statistiikkaa aina ohjelmaa kutsuttaessa annetun sekuntimäärän välein. Tämän hoitaa silmukka, joka nukkuu annetun sekuntimäärän verran ja tämän jälkeen laskee yhteen jokaisen Inserter-säikeen inserterCount-arvon. Tämän jälkeen lasketaan sisäänvientinopeus jakamalla inserterCount-arvojen summa annetulla sekuntimäärällä ja keskimääräinen sisäänvientinopeus jakamalla inserterCount-arvojen summien summa rivien sisään viemiseen sillä hetkellä kestäneellä ajalla. Nämä arvot sovellus tallentaa lokitiedostoon. Kun rivien sisään vienti on tehty, otetaan vielä keskiarvo koko sisäänvientiprosessin ajalta ja tallennetaan se omaan lokitiedostoonsa.

6.3 Testikoneet

Tietokantojen testaamista varten otettiin käyttöön kolme palvelinta. Tässä tutkimuksessa palvelimia kutsutaan numeroilla 21, 22 ja 23. Näistä numero 21 toimi koneena, josta dataa syöttävää sovellusta ajettiin. Numero 22 toimi koneena, jonne data tallennettiin ja numero 23 toimi koneena, josta dataa kyseltiin. Tällä kokoonpanolla saatiin simuloitua verkkotapahtumien keräysjärjestelmän kokoonpanoa, jossa on elementti, joka syöttää dataa tiedonkeräyspalvelimelle ja tiedonkeräyspalvelimen dataa kysellään ylemmällä kerroksella toimivalta välittäjäpalvelimelta. Koneina 21 ja 23 toimivat HP:n ProLiant DL360 G7 -koneet ja koneena 22 toimi HP:n ProLiant DL360 Gen9. Kaikille koneille asennettiin Red Hat Enterprise Linux versio 6.5. Koneilla oli seuraavanlaiset laitteistokokoonpanot:

Kone 21

- 2 * Intel(R) Xeon(R) CPU E5640 @ 2.67GHz, 4 ydintä + hyperthreading.
- 32 GiB muistia.
- 600 GiB kiintolevy käyttöjärjestelmälle.

Kone 22

- 2*Intel(R) Xeon(R) CPU E5-2667 v3 @ 3.20GHz, 8 ydintä + hyperthreading.
- 64 GiB keskusmuistia, 32 GiB kummallekin prosessorille.
- 600 GiB kiintolevy käyttöjärjestelmälle ja 7,2 TiB levypakka RAID 0+1 tilassa.

Kone 23

- Intel Xeon X5670 @ 2.93GHz, 6 ydintä + hyperthreading.
- 7.85 GiB muistia.
- 300 GiB kiintolevy käyttöjärjestelmälle.

Kaikkien koneiden välillä oli 1 Gbit verkkoyhteys. Parstream ja Cassandra asennettiin kummallekin koneelle. Parstreamista asennettiin versio 3.1.14 ja Cassandrasta Datastaxin Community-versio 2.1.4. Hyvät tietoturvaominaisuudet sisältävää Enterprise-versiota ei siis testattu, mutta Community-version pitäisi kuitenkin antaa kuva Cassandran suorituskyvystä.

6.4 Tietokantaklusterien toteuttaminen

Molemmille tietokannoille toteutettiin kahden tietokantasolmun klusteri edellä esitellyille koneille 22 ja 23. Klusteri toteutettiin siten, että kaikki data haluttiin viedä koneella 22 olevalle solmulle ja koneella 23 olevaa solmua haluttiin käyttää vain kyselyihin. Kuten kohdassa 5.8 esitettiin, tällaisen klusterin

toteuttaminen onnistuu kummallakin tietokannalla. Kummassakin tietokannassa pitää valita ”johtajasolmu”, jotta klusteri saadaan toimimaan. Kummankin tietokannan tapauksessa johtajasolmuna käytettiin konetta 23. Kummassakin tietokannassa datan tallentamisessa käytettiin koneella 22 kohdassa 9.3 mainittua 7,2 TiB kokoista levypakkaa.

Cassandrassa tietokantaklusterin kokoonpano ja lähes kaikki tietokannan asetukset voidaan määrittää cassandra.yaml-tiedostossa [Datastax, 2015d]. Cassandrassa johtajasolmua kutsutaan siemensolmuksi (seed node). Cassandra käyttää tietokantasolmujen välillä Gossip-protokollaa, jonka kautta solmut vaihtavat tietoja toisistaan ja oppivat klusterin tilan. Siemensolmulla on merkitystä erityisesti uuden solmun lisäämisen yhteydessä, sillä uusi solmu käyttää siemensolmua oppiakseen klusterin tilan.

Osioijaksi klusterissa valittiin ByteOrderedPartitioner, joka esiteltiin kohdassa 5.8. Nyt, kun tietokantaan viedään sarakeperhevarasto RRCRABTEST ja pääavaimen ensimmäiseksi osaksi int-tyyppinen inserter_id, hajauttaa Cassandra tällä pääavaimella tunnistettavan rivin sen mukaan, mikä inserter_id:n arvo on heksadesimaalimuodossa. Cassandra osaa määrittää klusterin arvoalueet automaattisesti tai ne voidaan määrittää käsin määrittämällä initial_token-arvo cassandra.yaml-tiedostossa. Tässä tapauksessa tehtiin niin, että koneelle 22 määritettiin init_valueksi 0 ja koneelle 23 sellainen arvo, että se on suurempi tai yhtä suuri kuin suurin käytetty inserter_id arvo. Tällä tavalla kaikki dataa sisään syöttävän sovelluksen raportit saatiin menemään koneelle 22. Koneella 22 concurrent_writes-muuttuja asetettiin suositeltuun arvoon 8*suoritinydinten määrä. Koneella 22 arvoksi tuli siis 128, koska ytimiä oli 16.

Parstreamissa tietokantaklusterin kokoonpano ja asetukset määritetään parstream.ini-nimisessä tiedostossa. Parstreamin asetuksissa solmulle 23 asetettiin rank-arvo 1 ja solmulle 22 rank-arvo 2. Näin saatiin toteutettua kahden solmun Parstream-klusteri, siten että solmusta 23 tehtiin johtajasolmu. Parstreamissa johtajasolmun vastuulla on informoida muita solmuja klusterin tilasta ja hallinnoida partitioiden yhdistelyä.

6.5 Datan syöttämisen tehokkuutta testaavat testit

Tietokantojen datan syötön suorituskykyä testattiin kohdassa 6.2 kuvatulla sovelluksella. Syöttönopeus on kaikkein kriittisin tekijä verkkotapahtumien keräysjärjestelmässä, sillä järjestelmään saattaa tulla todella suuria määriä tapahtumaraportteja yhden sekunnin aikana. Syöttönopeutta testattiin sekä 21-, että 22-koneella. Konetta 21 käytettäessä saatiin simuloitua aitoa tilannetta, jossa elementti lähettää dataa tiedonkeräyspalvelimelle verkon kautta. Koneella 22 taas saatiin testattua tilannetta, jossa

verkkoyhteyden nopeus ei muodostu hidasteeksi. Konella 22 on lisäksi vastaava laitteisto, jolla on saatu nykyisen käytössä olevan tietokannan testitulos 210 000 raporttia sekunnissa, kun käytetään 16 säiettä datageneraattorissa. Syöttönopeudessa oltiin kiinnostuneita ensinnäkin tietokantojen suorituskyvystä, kun dataa vietiin tietokantaan rinnakkain useammalla säikeellä. Kummassakin syöttötestissä Parstreamissa oli bittikarttaindeksi sarakkeella rrc_imsi. Tämä hidasti hieman Parstreamin syöttönopeutta. Testitulokset esittävissä taulukoissa nopeudet ovat keskiarvoja siltä ajalta kun dataa on syötetty.

6.6 Syöttötesti eri säiemäärillä koneella 21

Koneella 22 Cassandrassa datan viemistä Cassandraan ja Parstreamiin testattiin siten, että dataa vietiin sisään tietty aika, jonka jälkeen säikeiden määrää nostettiin yhdellä. Tätä toistettiin siten, että viimeiseksi dataa vietiin sisään 16 säikeellä. 16 säiettä testattiin siksi, että koneella 22 oli kaikkiaan 16 loogista suoritinydintä. Syöttämisessä esiintyi ongelmia erityisesti Cassandran kanssa, sillä kun säikeiden määriä kasvatettiin, alkoi syötön aikana tulla yhteysongelmia. Tästä syystä Cassandran tuloksista on listattu tulokset vain 12 säikeellä. Myös pitempiä ajoja tehtiin, mutta esimerkiksi 15 minuutin ajolla Cassandralla alkoivat yhteysongelmat jo 7 säikeen kanssa, joten tässä tapauksessa tulokset on nyt esitetty 5 minuutin ajolla.

Taulukko 4: Cassandran tulokset 5 minuutin ajolla.

Säikeet	Nopeus riviä/s
1	4058.0
2	5079.0
3	5114.0
4	4590.0
5	3968.0
6	3587.0
7	3015.0
8	2612.0
9	2417.0
10	2218.0
11	4417.0
12	2952.0

Taulukko 5: Parstreamin tulokset 5 minuutin ajolla.

Säikeet	Nopeus riviä/s
1	23278.0
2	43266.0
3	53934.0
4	76566.0
5	92852.0
6	104262.0
7	127333.0
8	134819.0
9	149166.0
10	137344.0
11	140819.0
12	146295.0
13	144819.0
14	145803.0
15	145311.0
16	146229.0

Kuten taulukoista 4 ja 5 huomataan, Cassandra ja Parstream käyttäytyvät erilailla, kun säikeiden määrää lisätään. Cassandrassa kahdella ja kolmella säikeellä saadaan hieman yhtä säiettä paremmat tulokset. Kun säikeiden määrää lisätään siitä ylöspäin, vientinopeus hidastuu. Parstreamilla säikeiden

kasvattaminen nostaa vientinopeutta selvästi ja nopeutta on saatu lisää kasvattamalla säikeiden määrää aina 9 säikeeseen asti. Kumpikin kanta käyttäytyi samalla tavalla myös pidemmillä ajoilla, joten taulukoiden tuloksista saa kuvan tietokantojen käyttäytymisestä eri säiemäärillä. Parstreamilla nopeus hidastuu 9 säikeen jälkeen todennäköisesti siksi, että käytössä on vain 8 fyysistä ydintä, joten vain 8:n säiettä voidaan ajaa aidosti rinnakkain. Ilmeisesti prosessorin HyperThreading-ominaisuus, joka lisää prosessoriin virtuaalisia ytimiä, antaa hieman lisänopeutta, kun säikeitä on yli 8.

Kun dataa ajaa Parstreamiin sisään maksiminopeudella ja verkon käyttöastetta katsoo iftop-sovelluksella, dataa liikkuu verkon läpi noin 550–600 Mbit sekunnissa. Koska verkon suorituskyky on 1 Gbit/s, verkon nopeus ei muodostu tässä tapauksessa hidasteeksi. 550–600 Mbit sekunnissa vastaa varsin hyvin teoreettista nopeutta mikäli nopeus lasketaan Parstreamin taulun koon perusteella. Liittessä 2 kuvatussa taulussa on 14 8-bittistä unsigned int -saraketta, 55 16-bittistä unsigned int -saraketta, 20 32-bittistä unsigned int -saraketta, 1 64-bittinen timestamp-kenttä, 5 16-merkin pituista varstring-merkkijonoa, 1 40-merkin pituinen varstring-merkkijono ja 1 100-merkin pituinen varstring-merkkijono. Jos varstring ajatellaan jonona 8-bittisiä char-tyyppejä, on yhden rivin koko 3930 bittiä. Kun rivejä viedään sisään Parstreamiin maksiminopeudella 149166 riviä sekunnissa, dataa pitäisi virrata 3930b*149166.0=586222380b=0.5459621362388l3GBit sekunnissa. Tämä vastaa varsin hyvin iftop-sovelluksella mitattua lukemaa, joten Parstreamin Streaming Import -rajapinta ei näytä tuovan juuri lisäkuormaa datan siirtoon.

6.7 Syöttötesti eri säiemäärillä koneella 22

Koneella 22 verkkoyhteyden nopeus ei voi muodostua hidasteeksi, sillä tietokanta on samalla palvelimella kuin dataa syöttävä sovellus. Kone 22 on myös nopeampi kuin kone 21 ja samantasoisella laitteistolla on myös aikaisemmin testattu nykyisen tietokannan nopeutta, joka on 210 000 raporttia sekunnissa. Paikallisesti koneella 22 testattuna voidaan vertailla Cassandran ja Parstreamin nopeutta nykyisin käytössä olevaan tietokantaan.

Koneella 22 Cassandrassa datan viemiseen käytettiin isoimmillaan 12 säiettä, koska suuremmalla säiemäärällä tuli yhteysongelmia. Parstreamissa käytettiin enimmillään 16 säiettä. Isompiakin määriä testattiin, mutta nopeus ei enää juurikaan noussut. Cassandran tulokset on listattu taulukossa 6 ja Parstreamin taulukossa 7.

Taulukko 6: Cassandran tulokset 5 minuutin ajolla.

Säikeet	Nopeus riviä/s
1	1132.0
2	5454.0
3	4918.0
4	4491.0
5	4032.0
6	3619.0
7	3161.0
8	2885.0
9	2492.0
10	2187.0
11	2063.0
12	2029.0

Taulukko 7: Parstreamin tulokset 5 minuutin ajolla.

Säikeet	Nopeus riviä/s
1	30459.0
2	56393.0
3	79114.0
4	101180.0
5	117278.0
6	130448.0
7	151606.0
8	167868.0
9	197233.0
10	220393.0
11	242622.0
12	241366.0
13	248032.0
14	256098.0
15	272574.0
16	273633.0

Koneella 22 Parstreamin syöttönopeus on selvästi korkeampi kuin koneella 22. Syöttönopeus on myös 16 säiettä käytettäessä suurempi kuin nykyisin käytössä olevan tietokannan 210 000 raporttia sekunnissa. Kuten edellisessä kohdassa mainittiin, verkkoyhteyden nopeus ei muodostunut hidasteeksi, joten korkeampi nopeus koneella 22 johtuu todennäköisesti koneen paremmasta laitteistosta. Käytössä on kaksi 8-ytimistä prosessoria, joten nopeus kasvaa aina 16 säikeeseen asti. Cassandran

syöttönopeuden käyttäytyminen on suurin piirtein sama kuin koneella 21, joten 22-koneen parempi suorituskyky ei paranna Cassandran syöttönopeutta.

Cassandran selvästi heikompi suorituskyky Parstreamiin verrattuna johti siihen, että Cassandraa päätettiin tutkia hieman enemmän muuttamalla eri asetuksia cassandra.yaml-tiedostossa. Cassandran asetuksia testattiin ainakin asettamalla durable_writes-muuttuja off-tilaan, jolloin dataa kirjoitettaessa Cassandran commitlog-ohitetaan. Cassandra siis kirjoittaa datan ensin muistissa olevaan memtableen ja commitlog-tiedostoon ennen datan vientiä lopulliseen data-hakemistoon, jossa tietokantaa säilytetään. Durable_writes-muuttujan asettamisella off-tilaan ei kuitenkaan ollut juurikaan vaikutusta kirjoitusnopeuteen. Myös memtable_heap_space_in_mb- ja memtable_offheap_space_in_mb-muuttujia muutettiin suuremmaksi. Näin muistissa sijaitseva memtable-rakenne, johon data kirjoitetaan Cassandrassa ennen sen syöttämistä kantaan, olisi suuremman kokoinen ja näin saataisiin mahdollisesti lisänopeutta. Tämäkään ei kuitenkaan lisännyt kirjoitusnopeutta.

Cassandran syöttönopeudessa päätettiin vertailla myös eroja executeAsync()- ja execute()-metodien väleillä ja vertailla tilanteeseen, jossa nämä metodit on kommentoitu pois ja kantaan ei viedä mitään. Testit tehtiin minuutin ajolla ja maksimissaan 14 säikeellä. Tulokset nähdään taulukossa 8.

Taulukko 8: Sisäänvientitesti Cassandraan minuutin ajolla per säie.

executeAsync()			execute()			Ei datan sisään vientiä	
Säikeet	Nopeus riviä/s		Threads	Nopeus riviä/s		Threads	Nopeus riviä/s
1	5333.0		1	3384.0		1	95076.0
2	5846.0		2	5538.0		2	201692.0
3	5538.0		3	6461.0		3	298153.0
4	5142.0		4	6153.0		4	389230.0
5	5000.0		5	5714.0		5	494461.0
6	4615.0		6	5538.0		6	569538.0
7	4000.0		7	5384.0		7	659384.0
8	4000.0		8	4923.0		8	788769.0
9	3600.0		9	4800.0		9	860769.0
10	3157.0		10	4615.0		10	922307.0
11	2933.0		11	4400.0		11	1039384.0
12	2526.0		12	4235.0		12	987076.0
13	2476.0		13	4000.0		13	850923.0
14	4315.0		14	4000.0		14	1306923.0

Taulukon 8 tuloksista huomataan, että execute()-metodia käytettäessä nopeus on jonkin verran executeAsync()-metodia korkeampi, kun säikeiden määrää lisätään. Merkittävää nopeuseroa ei kuitenkaan ole ja sisäänvientinopeuden käytös näyttää suurin piirtein samalta kummassakin tapauksessa. Kun execute()-metodia, eikä executeAsync()-metodia ei käytetä, sisäänvientinopeus näyttää kasvavan tasaisesti kuitenkin pysähtyen 11-säikeen kohdalla. Koska nopeus kasvaa säikeitä kasvattaessa, Cassandran sisäänvientinopeuden lasku ei johdu InserterDriver-sovelluksesta, vaan execute()- ja execute_async()-metodien käytöstä. Näin ollen sisäänvientinopeuden lasku johtuu joko Java-ajurista tai tietokannasta itsestään.

Cassandrassa taulun pääavaimen määrityksellä on paljon merkitystä tiedon hajautuksen ja tietokannan suorituskyvyn suhteen. Mitä enemmän sarakkeita pääavaimen määritykseen laittaa, sitä hitaammin dataa menee sisään. Testejä tehtiin aikaisemmin mainitulla RRCRABTEST-sarakeperhevarastolla siten, että sen pääavaimen määritystä muutettiin ja tulokset eri pääavainyhdistelmillä on listattu taulukossa 9.

Taulukko 9: Sisäänvientinopeus Cassandrassa eri pääavaimilla.

Pääavain	Nopeus r/s
Rowindex	11428.0
Inserter_id, rowindex	7272.0
Inserter_id, report_time, rowindex	5714

Nopeimmillaan Cassandra on siis, jos käytetään vain yhtä saraketta pääavaimen osana. Tällöin Cassandran sarakeperhevarastossa käytetään pelkkää riviavainta tunnistamaan rivit Cassandran sarakeperhevarastossa. Tässä tulee ongelmaksi se, että riviavaimen pitäisi olla aina uniikki ja tämä rajoittaa kyselymahdollisuuksia Cassandrassa. Esimerkiksi, jos halutaan hakea dataa tietyiltä arvoväleiltä, Cassandra edellyttää, että sarake on pääavainmäärityksen osana.

Verkkotapahtumien keräysjärjestelmässä käyttäjiä kiinnostaa erityisesti tietyltä aikajaksolta tehdyt kyselyt, joten pääavaimessa pitäisi olla ainakin aikaleima. Näin ollen, jos tässä tutkimuksessa määritetty järjestelmä haluttaisiin toteuttaa, pitäisi sarakeperhevaraston pääavainmääritys sisältää sarakkeet inserter_id, report_time. Tämä määritys on kuitenkin ongelmallinen siitä syystä, että report_time-sarakkeessa oleva aikaleima saattaa olla sama. Tätä tilannetta simuloidaan InserterDriver-sovelluksessa päivittämällä aikaleima joka 15 syötön välein. Koska aikaleimat eivät ole uniikkeja, pääavainpari inserter_id ja report_time ei riitä rivien tunnistamiseen Cassandrassa, sillä tällä pääavainparilla Cassandra tallentaa saman inserter_id ja report_time parin sisältävistä RRC/RAB-raporteista vain viimeisimmän. Näin ollen pääavaimeen pitäisi lisätä myös kolmas sarake, jotta kaikki raportit saataisiin vietyä kantaan.

6.8 Kyselytestit

Tietokantakyselyissä käyttäjää kiinnostavat paitsi hakea raportteja tietyltä aikaväliltä, myös hakea jokin tietty raportti, tietyn sarakkeen arvon perusteella. Kiinnostavia sarakkeita ovat ainakin IMSI, IMEI, MSISDN, URL-osoite, IPv4-osoite ja IPv6-osoite. Tässä kohdassa on tutkittu tietokantojen suorituskykyä suurilla rivimäärillä. Tietokantakyselyitä kokeiltiin Cassandran cqlsh-kyselysovelluksen avulla ja Parstreamin kyselyitä Parstreamin pnc-sovelluksen avulla. Testit tehtiin 23-koneelta.

Cassandran CQL-kyselyt ovat varsin rajallisia, jos vertaa SQL-kyselyihin. WHERE-kyselyitä ei voi tehdä ellei saraketta ole indeksoitu. Tämä tarkoittaa sitä, että sarakkeen pitää olla joko osana sarakeperheen pääavainmääritystä tai sarakkeeseen täytyy luoda toissijainen indeksi (eng. secondary index). Cassandrassa toissijaisen indeksin käyttö on kuitenkin rajallista, sillä se sallii vain WHERE-kyselyt, jossa haetaan jotain tiettyä arvoa (esimerkiksi SELECT * FROM RRCRABTEST WHERE

rowindex=1). Hakua joltakin arvoalueelta ei voi kuitenkaan tehdä. Esimerkiksi, jos edellä mainitussa lauseessa määriteltäisiin WHERE rowindex>1, tulisi virhe. Sarakeperheen indeksointi on siis suunniteltava sen mukaan, mitä kyselyitä tietokantaan halutaan tehdä.

Kyselyitä testattiin Cassandrassa ja Parstreamissa viemällä suuria määriä rivejä kantaan InserterDriver-sovelluksella. Testejä tehtiin eri rivimäärillä. Suurimmillaan testattiin 1 000 000 000 riviä kummassakin kannassa. Kyselytestien tulokset erosivat suuresti. Cassandrassa rivien lukumäärän hakeva SELECT COUNT(*) FROM RRCRABTEST -kysely oli todella hidas Parstreamiin verrattuna. 500 000 rivillä kokeiltuna kysely kesti 3 min 28 sekuntia ja 1 000 000 rivillä 10 min 56 sekuntia. Tästä isommilla määrillä kokeileminen meni hankalaksi, sillä kyselyn suorittamiseen kului liikaa aikaa. Parstreamilla vastaavaan kyselyyn meni 0,059 sekuntia 1 000 000 rivillä ja 100 000 000 rivillä aikaa kului noin 2,742 sekuntia.

Kun RRCRABTEST-sarakeperhettä testattiin 1 000 000 000 rivillä, huomattiin, että Cassandran sarakeperheen statistiikkoja listaavan cfhistogramsin arvoissa tapahtui ylivuoto. Partition Size- ja Cell Count -arvot olivat tällä rivimäärällä kasvaneet liian suuriksi. Tästä johtuen kyselyissä tapahtui Cassandrassa virheitä ja kyselyjä ei saatu näillä rivimäärillä tehtyä. 10 000 000 ja 100 000 000 rivillä Cassandra pysyi kuitenkin vakaana ja tällä rivimäärällä saatiin tehtyä kyselytestejä. Mielenkiintoisena havaintona Parstreamiin nähden oli, että SELECT * FROM RRCRABTEST -kysely palautti tuloksen Cassandrassa heti, mutta Parstreamissa rivien hakemiseen paljon aikaa. Cassandrassa ison tulosjoukon esittäminen on toteutettu siten, että ruudulle haetaan vain tietty rivimäärä kerrallaan ja tätä rivimäärää vieritetään alaspäin. Parstreamissa taas tulosjoukko haetaan kokonaisuudessaan ruudulle. Parstreamilla tuli tässä ongelmia myös puskurin koon kanssa, kun yli gigatavun kokoinen tulosjoukko yritettiin hakea.

Eräs mielenkiintoisista kyselyistä oli tietyn IMSI-numerosarjan haku tietokannasta. Cassandrassa tätä varten tehtiin ensin sarakeperhe siten, että PRIMARY KEY -määritykseen asetettiin sarakkeet inserter_id, report_time ja rrc_imsi. Kun tällä tavalla määritettyyn tauluun tekee kyselyitä, on otettava huomioon, että kyselyä, jossa WHERE-lauseessa kysellään pelkkää IMSI-numerosarjaa, ei voi tehdä. Kyselyssä pitää määrittää myös aikaleima report_time siltä riviltä, jolta IMSI-numerosarjaa kysellään. Report_time-aikaleimaa voi tässä tapauksessa vertailla vain yhtäsuuruusmerkillä. Pienempi kuin ja suurempi kuin vertailut eivät ole mahdollisia. Ei ole siis mahdollista tehdä hakua, jossa haetaan IMSI-numerosarja tietyltä aikaväliltä, vaan tietty IMSI pitää hakea tietyltä ajalta.

Haku SELECT * FROM RRCRABTEST WHERE report_time=’2015-04-01 20:00:00’ AND rrc_imsi=”404100223808183” ei kuitenkaan tuota yhtään tulosta Cassandraan, vaikka kannassa löytyy report_time ja rrc_imsi samassa annetussa muodossa kuin hakulauseessa. CQL 3.2.0 dokumentaation mukaan Cassandra tallentaa aikaleimat millisekunteina epookista 1.1.1970 00:00:00 [CQL, 2015]. Ilmeisesti vertailua tehdessä annettu aikaleima ja tallennettu aika eivät vastaa toisiaan millisekunteina ja näin Cassandra ei näe yhtäläisyyttä aikaleimoissa.

Cassandraa päätettiin testata siten, että pääavaimeksi asetettiin yhdistelmä inserter_id, rowindex ja rrc_imsi. Näin määritettyyn tauluun vietiin 100 000 000 riviä. Tällä määrällä avaimen saa kyseltyä nopeasti, mikäli tietää oikean inserter_id, rowindex ja rrc_imsi -yhdistelmän. Haun tulos saadaan heti. Cassandra on siis nopea, mikäli tietää mitä hakee. Tämä johtuu Cassandran avain-arvo-tyyppisestä rakenteesta. Muuten kyseleminen Cassandrasta on varsin hankalaa, sillä kyselymahdollisuudet ovat hyvin rajallisia verrattuna perinteiseen tietokantaan.

Parstreamissa kyselyitä nopeuttaa bittikarttaindeksointi. Vastaavanlainen IMSI-kysely voidaan tehdä Parstreamissakin varsin nopeasti, mikäli kyseltävät sarakkeet on bittikarttaindeksoitu. Kun testitaulussa inserter_id, report_time ja rrc_imsi on indeksoitu, 100 000 000 rivillä Cassandraa vastaavan kyselyn tekeminen kesti noin 3 sekuntia. Mikäli indeksointia ei ole, kyselyssä kestää kauan. Esimerkiksi kyselyssä SELECT * FROM rrcrabtest WHERE inserter_id=14 and rowindex=100 kestää 497.043 sekuntia.

7 Loppupäätelmät

Tehdyissä tutkimuksissa Parstream osoittautui suorituskyvyltään selvästi Cassandraa paremmaksi, kun datan syöttämistä testattiin eri määrillä säikeitä. Myös kyselyissä Parstream oli vahvoilla. Cassandrassa kyselyjen määrä on paljon rajoitetumpi verrattuna Parstreamiin. Sarakeperhevaraston pääavain pitää suunnitella sen mukaan, mitä kyselyitä haluaa tehdä. Cassandralla saa kyllä tehtyä nopeita kyselyitä, jos tietää sen pääavaimen arvojen yhdistelmän, jolla dataa kysellään. Toisaalta Parstreamista löytyy bittikarttaindeksointiominaisuus, jolla nopeita kyselyitä saa myös tehtyä. Lisäksi Parstreamissa monet perinteiset SQL-kyselyt ovat mahdollisia toisin kuin Cassandran CQL-kielessä, jonka kyselymahdollisuudet ovat rajoitetumpia. Parstreamilla verkkotepahtumien keräysjärjestelmässä tehtävät kyselyt saataisiin helpommin toteutettua. Tutkimuksessa datan tallentamiseen käytettiin vain yhtä solmua. Datastaxin, [2013] tekemässä tutkimuksessa Cassandra oli erityisen vahvoilla, kun solmujen määrää lisättiin. Jatkotutkimuksen kannalta voisi olla mielenkiintoista verrata Parstreamia ja Cassandraa, kun datan tallentamiseen käytetään useampaa solmua. Parstream on kuitenkin yhdelläkin solmulla niin paljon nopeampi, että todennäköisesti Parstream pärjäisi hyvin Cassandraa vastaan useamman solmun testeissä.

NoSQL-tietokannat eivät ylipäätään sovellu kovinkaan hyvin verkkotapahtumien keräysjärjestelmän tietokannaksi, sillä monet niistä on suunniteltu web-sivustoja varten ja sellaiselle datalle, johon tehdään paljon avain-arvo-pohjaisia kyselyitä. Verkkotapahtumien keräysjärjestelmässä olisi tärkeää pystyä tekemään hakuja tietyiltä ajanjaksoilta ja useiden eri sarakkeiden perusteella. Avain-arvo-varastoissa käytettävä tietomalli on liian yksinkertainen vastaamaan näihin vaatimuksiin. Toisaalta joissakin avain-arvo-varastoissa on kehittyneitä ominaisuuksia. Esimerkiksi BerkeleyDB tukee Grolingerin et al., [2013] mukaan SQLite-kirjastoa, jonka avulla SQL-kyselyt olisivat mahdollisia. Tietokanta on sulautettu tietokanta, niin kuin nykyinenkin käytössä oleva tietokanta ja tukee datatiedostojen kryptausta. Tämä tietokanta voisi olla mielenkiintoinen jatkotutkimuksen kannalta. Dokumenttivarastoissa tieto tallennetaan JSON-formaatissa ja tässä muodossa tallennettuna testeissä käytetty RRC/RAB-raportti veisi paljon tilaa. Graafitietokannat on taas suunniteltu paljon suhteita sisältävälle datalle ja tätä verkkotapahtumien keräysjärjestelmässä ei juuri ole. Tietotyyppien tuki on NoSQL-tietokannoissa SQL-tietokantoja rajoitetumpi, eikä NoSQL-tietokannoissa ole mitään standardia rajapintaa, kuten JDBC ja ODBC.

NewSQL-tietokannoissa NuoDB:tä tutkittiin hieman Cassandran ja Parstreamin ohella. NuoDB:lle syöttötestiä tehdessä saatiin nopeudeksi 48307 raporttia sekunnissa HP:n ProLiant DL360 Gen9 -laitteistoa käyttävällä koneella. Tietokanta oli siis nopeampi kuin Cassandra, mutta hitaampi kuin

Parstream. Lisäksi, kuten kohdassa 5.5 mainittiin, Grolingerin et al. [2013] mukaan NuoDB:ssä datan hajautus ei ole käyttäjän hallittavissa. NewSQL-tietokannoissa löytyy paljon mielenkiintoisia vaihtoehtoja ja esimerkiksi JustOneDB:tä on harkittu tutkittavaksi yrityksessä. NewSQL-tietokannoissa hyvänä puolena on tuki SQL-kyselykielelle ja tietotyypeille. Lisäksi myös ODBC:tä ja JDBC:tä tuetaan monissa tietokannoissa.

Yhteenvetona tästä tutkimuksesta on, että Parstream on testien mukaan selvästi soveltuvin valinta hajautusominaisuuksiensa ja suorituskykynsä puolesta verkkotapahtumien keräysjärjestelmän tietokannaksi. Turvallisuusominaisuuksissa Parstreamissa on puutteita ja turvallisuusvaatimukset pitäisi ratkaista muilla keinoin kuin turvautumalla tietokannan ominaisuuksiin. Tosin NoSQL- ja NewSQL-tietokannoissa turvallisuusominaisuudet ovat ylipäätänsäkin puuttellisia, joten ihan täydellisesti vaatimuksia vastaavaa tietokantaa ei näistä löydy.

Viiteluettelo

[917/2014, 2014] Tietoyhteiskuntakaari (917/2014). 2014. Päivitetty 7.11.2014.

[2002/58/EY, 2002] Direktiivi 2002/58/EY, Henkilötietojen käsittely ja yksityisyyden suoja sähköisen viestinnän alalla (sähköisen viestinnän tietosuojadirektiivi), annettu 12.7.2002.

[Abramova et al, 2014] Abramova, V., Bernardino, J. and Furtado, P. (2014). Which NoSQL Database? A Performance Overview. *Open Journal of Databases (OJDB)*, **1**(2), 17-24.

[Accumulo, 2015] Accumulo-tietokanta. https://accumulo.apache.org/, 2015. Viitattu 29.4.2015.

[Aerospike, 2015] Aerospike-tietokanta. http://www.aerospike.com/, 2015. Viitattu 13.5.2015.

[AllegroGraph, 2015] AllegroGraph-tietokanta. http://franz.com/agraph/allegrograph/, 2015. Viitattu 14.5.2015.

[Arvonen, 2003] Arvonen, T., 2003. Reaaliaikaisten avaintehokkuustekijöiden käsittely Nokia NetActTM ympäristössä. Diplomityö. Tampere: Tampereen teknillinen yliopisto.

[Aslett, 2011] Aslett, M. How will the database incumbents respond to NoSQL and NewSQL?, 2011.

[Astrahan et al., 1976] Astrahan, M. M., Blasgen, W. D., Chamberlin, D., Eswaran, K. P., Gray J. N., Griffiths, P.P., King, W.F., Lorie, R.A., McJones, P.R., Mehl, J.W., Putzolu. G.R., Traiger, I. L., Wade, B.W. and Watson, V., 1976, System R: Relational approach to database management. *ACM Trans. Database Syst. 1, 2* (June 1976), 97-137.

[Baker, 1992] Baker, H.G. Relational databases considered harmful (relative to object-oriented databases.) ACM Forum. *Comm. of the ACM* **35**, 4 (April 1992), 16, 18.

[BerkeleyDB, 2015] Oracle Berkeley DB 12c. http://www.oracle.com/technetwork/database/database-technologies/berkeleydb/overview/index.html, Viitattu 14.5.2015.

[Bernstein et al., 1987] Bernstein, P.A., Hadzilacos V. and Goodman, N. (1987): *Concurrency control and recovery in database systems* (Vol. 370). New York: Addison-Wesley.

[Bienert et al., 2013] US 20130204850 A1, 2015. Method and system for compressing data records and for processing compressed data records, empulse. GmbH/Parstream Gmbh, Saksa. (Jörg Bienert, Michael Hummel, Norbert Heussler), US 13/576,082, 4.2.2011. Julk. 8.8.2013. 7 p.

[Bray, 2014] Bray, T. RFC 7159–The JavaScript object notation (JSON) data interchange format, 2014.

[Brewer, 2000] Brewer, A., Towards Robust Distributed Systems, 2000. Saatavilla osoitteesta https://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf. Viitattu 28.4.2015.

[BSON, 2014] Binary JSON -spesifikaatio. http://bsonspec.org/, 2015. Viitattu 3.1.2015.

[Cassandra, 2015] Apache Cassandra -tietokanta. http://cassandra.apache.org/, 2015. Viitattu 3.1.2015.

[Cattell and Barry, 2000] Eastman, J., Jordan, D., Russell, C., Schadow, O., Stanienda, T. and Velez, F. (2000). *The object data standard: ODMG 3.0 (Vol. 1)*. R. G. G. Cattell, & D. K. Barry (Eds.). San Francisco: Morgan Kaufmann.

[Chamberlin et al., 1974] Chamberlin, D. D. and Boyce, R.. SEQUEL: A Structured English Query Language. *Proceedings of the 1974 ACM SIGFIDET Workshop*

[Chang et al., 2006] Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A. and Gruber, R.E. (2006): Bigtable: A Distributed Storage System for Structured Data. *ACM Trans. Comput. Syst.* **26**, 2, Article 4 (June 2008), 26 pages.

[Clustrix, 2015] Clustrix-tietokanta. http://www.clustrix.com/, 2015. Viitattu 14.5.2015.

[CODASYL, 1971] CODASYL Data Base Task Group 1971. Report of the CODASYL Data Base Task Group, Assoc. for Computing Machinery, N.Y

[Codd, 1970] Codd, E., A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM* **13** (6): 377.

[Cooper et al., 2010] Cooper, B., Silberstein, A., Tam, E., Ramakrishnan, R. and Sears, R.: Benchmarking cloud serving systems with YCSB. *In Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC '10). ACM*, New York, NY, USA, 143-154, 2010.

[Corbett el al., 2013] Corbett, J.C., Dean, J., Epstein, M., Fikes, A., Frost, C., Furman, J.J. and Woodford, D. (2013). Spanner: Google's globally distributed database. *ACM Transactions on Computer Systems (TOCS)*, **31**(3), 8.

[Couchbase, 2011] Couchbase, UnQL Query Language Unveiled by Couchbase and SQLite. http://www.couchbase.com/press-releases/unql-query-language, 2011. Viitattu 24.2.2015.

[Couchbase, 2015] Couchbase-tietokanta. http://www.couchbase.com, 2015. Viitattu 14.5.2015.

[CouchDB, 2015] CouchDB-tietokanta. http://couchdb.apache.org/, 2015. Viitattu 14.5.2015.

[CQL, 2015] Cassandra Query Language (CQL) v3.2.0. http://cassandra.apache.org/doc/cql3/CQL.html#usingdates, 2015. Viitattu 29.4.2015.

[Datastax, 2013] Datastax, Benchmarking Top NoSQL Databases A Performance Comparison for Architects and IT Managers. White Paper By Datastax Corporation, February 2013.

[Datastax, 2015a] Apache Cassandra 2.1 Documentation. Partitioners. http://docs.datastax.com/en/cassandra/2.1/cassandra/architecture/architecturePartitionerAbout_c.html, 2015. Viitattu 29.5.2015.

[Datastax, 2015b] CQL for Cassandra 1.2 Data types. http://www.datastax.com/documentation/cql/3.0/cql/cql_reference/cql_data_types_c.html, 2015. Viitattu 3.1.2015.

[Datastax, 2015c] Java driver 2.1 for Apache Cassandra. http://www.datastax.com/documentation/developer/java-driver/2.1/common/drivers/introduction/introArchOverview_c.html, 2015. Viitattu 29.4.2015.

[Datastax, 2015d] Datastax Documentation Apache Cassandra™ 2.1. The cassandra.yaml configuration file. http://docs.datastax.com/en/cassandra/2.1/cassandra/configuration/configCassandra_yaml_r.html, 2015. Viitattu 29.4.2015

[DB2, 2015] IBM DB2 database software. http://www-01.ibm.com/software/data/db2/, 2015. Viitattu 15.5.2015.

[DB-Engines, 2015a] DB-Engines: Knowledge Base of Relational and NoSQL Database Management Systems. http://db-engines.com/en/, 2015. Viitattu 1.5.2015.

[DB-Engines, 2015b] DB-Engines Ranking of Wide Column Stores. http://db-engines.com/en/ranking/wide+column+store, 2014. Viitattu 3.1.2015.

[dbShards, 2015] dbShards-tietokantaratkaisu. http://dbshards.com/, 2015. Viitattu 15.5.2015.

[Dean and Ghemawat, 2008] Dean, J. and Ghemawat, S., MapReduce: simplified data processing on large clusters. *Communications of the ACM* **51**(1) (2008): 107-113.

[Dipperstein, 2014] Dipperstein, M., Run Length Encoding (RLE) Discussion and Implementation, 2014. Saatavilla osoitteessa: http://michael.dipperstein.com/rle. Viitattu 20.2.2014.

[DynamoDB, 2015] DynamoDB-tietokanta. http://aws.amazon.com/dynamodb/, 2015. Viitattu 14.5.2015.

[Edlich, 2009] Edlich, S., NoSQL archive page. http://nosql-database.org/, 2009. Viitattu 29.4.2015.

[Ehcache, 2015] Ehcache-tietokanta. http://ehcache.org/, 2015. Viitattu 13.5.2015.

[Ehringer, 2014] Ehinger, B., MySQL vs PostgreSQL. http://itxdesign.com/mysql-vs-postgresql/, 2014. Viitattu 29.4.2015.

[Elmasri and Navathe, 1989] Elmasri, R. and Navathe, S., *Fundamentals of the Database Systems,* The Benjamin/Cummings Publising Company Inc, 1989

[Elmasri and Navathe, 2011] Elmasri, R. and Navathe, S., *Fundamentals of the Database Systems (6th Edition),* Addison-Wesley, 2011

[Evans, 2008] Evans, E., Kirjoitus grokbase.com:ssa, http://grokbase.com/t/cassandra/user/1162fkpwx2/release-0-8-0, 2008. Viitattu 29.4.2015.

[Evans, 2009] Evans, E., Evans's Weblog, NOSQL 2009. http://blog.sym-link.com/2009/05/12/nosql_2009.html, 2009. Viitattu 29.4.2015.

[Fortune, 2014] Fortune, S., A Brief History of Databases. http://avant.org/media/history-of-databases, 2014. Viitattu 29.4.2015.

[FoundationDB, 2015] FoundationDB-tietokanta. https://foundationdb.com/, 2015. Viitattu 15.5.2015.

[Geiger, 1995] Geiger, K., *Inside ODBC*. Microsoft Press, 1995.

[GenieDB, 2015] GenieDB. http://www.geniedb.com/, 2015. Viitattu 15.5.2015.

[Grolinger et al., 2013] Grolinger, K., Higashino, W., A. and Tiwari, A., Data management in cloud environments: NoSQL and NewSQL data stores. *Journal of Cloud Computing: Advances, Systems and Applications 2013*, **2**:22.

[Haerder and Reuter, 1983] Haerder, T. and Reuter, A., Principles of transaction-oriented database recovery. *ACM Comput. Surv.* **15**, 4 (December 1983), 287-317.

[HBase, 2015] Apache HBase -tietokanta. http://hbase.apache.org/, 2015. Viitattu 29.4.2015.

[Hecht and Jablonski, 2011] Hecht, R. and Jablonski, S., NoSQL evaluation: A use case oriented survey. *International Conference on Cloud and Service Computing (CSC'11)*, Hong Kong, Kiina, 2011, sivut 336-341.

[HyperGraphDB, 2010] HyperGraphDB-tietokanta. http://www.hypergraphdb.org/, 2010. Viitattu 14.5.2015.

[Hypertable, 2015] Hypertable-tietokanta. http://hypertable.org/, 2015. Viitattu 29.4.2015.

[InfiniDB, 2015] GitHub: InfiniDB Data Warehouse. https://github.com/infinidb/infinidb, 2015. Viitattu 15.5.2015.

[ISO/IEC 9075, 1992] Information technology -- Database languages – SQL, International Organization for Standardization, Geneve, Sveitsi. Saatavilla osoitteesta http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=16663. Viitattu 18.2.2015.

[ISO/IEC 9075(4,6), 1999] Information technology -- Database languages -- SQL – (Part 4, Part 6), International Organization for Standardization, Geneve, Sveitsi. Viitattu 18.2.2015.

[ISO/IEC 9075(4, 14), 2003] Information technology -- Database languages -- SQL – (Part 4, Part 14), International Organization for Standardization, Geneve, Sveitsi. Viitattu 18.2.2015.

[ISO/IEC 9075(14), 2006] Information technology -- Database languages -- SQL -- Part 14: XML-Related Specifications, International Organization for Standardization, Geneve, Sveitsi. Saatavilla osoitteesta http://www.iso.org/iso/iso_catalogue/catalogue_ics/catalogue_detail_ics.htm?csnumber=38647. Viitattu 18.2.2015.

[ISO/IEC 9075(4), 2008] Information technology -- Database languages -- SQL -- Part 4: Persistent Stored Modules (SQL/PSM), International Organization for Standardization, Geneve, Sveitsi. Viitattu 18.2.2015.

[ISO/IEC 9075(4), 2011] Information technology -- Database languages -- SQL -- Part 4: Persistent Stored Modules (SQL/PSM), Geneve, Sveitsi. Saatavilla osoitteesta http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=53684. Viitattu 18.2.2015.

[JustOneDB, 2015] JustOneDB-tietokanta. http://www.justonedb.com/, 2015. Viitattu 15.5.2015.

[Kadebu and Mapanga, 2014] Kadebu, P. and Mapanga, I., A Security Requirements Perspective towards a Secured NOSQL Database Environment. *International Journal of Advance Research and Innovation, Institution of Engineers*, Delhi State Center,India, 2014

[Klein et al., 2015] Klein, J., Gorton, I., Ernst, N., Donohoe, P., Pham, K. and Matser, C., Performance Evaluation of NoSQL Databases: A Case Study. In: *Proceedings of the 1st Workshop on Performance Analysis of Big Data Systems (PABS '15)*. ACM, New York, NY, USA, 2015, 5-10.

[Klishin and Monadovič, 2015] Klishin, M., Monadovič, A.P., ClojureWerkz, Cassandra Data Modeling. Saatavilla osoitteesta: http://clojurecassandra.info/articles/data_modelling.html. Viitattu 29.4.2015.

[Li and Manoharan, 2013] Li, Y. and Manoharan, S., A performance comparison of SQL and NoSQL databases. *Communications, Computers and Signal Processing (PACRIM), 2013 IEEE Pacific Rim Conference on* , vol., no., pp.15,19, 27-29 Aug. 2013

[Mäki, 2008] Mäki K., 2008. Käyttäjäpohjaisen tiedon tehokas välittäminen ja suodattaminen verkonvalvontajärjestelmässä. Diplomityö. Tampere: Tampereen teknillinen yliopisto.

[MarkLogic, 2015] MarkLogic-tietokanta. http://www.marklogic.com/, 2015. Viitattu 15.5.2015.

[Memcached, 2009] Memcached-tietokanta. http://memcached.org/, 2009. Viitattu 13.5.2015.

[MemSQL, 2015] MemSQL-tietokanta. http://www.memsql.com/, 2015. Viitattu 15.5.2015.

[MongoDB, 2015] MongoDB-tietokanta. http://www.mongodb.com/, 2015. Viitattu 3.1.2015.

[MS SQL Server, 2015] Microsoft SQL Server. http://www.microsoft.com/en-us/server-cloud/products/sql-server/, 2015. Viitattu 15.5.2015.

[MySQL, 2015] MySQL-tietokanta. https://www.mysql.com/, 2015. Viitattu 15.5.2015.

[Nelubin and Engber, 2013] Nelubin, D. and Engber, B., Ultra-High Performance NoSQL Benchmarking: Analyzing Durability and Performance Tradeoffs. Thumbtack Technology, Inc., White Paper, 2013.

[Neo4J, 2014] Neo4J-tietokanta. http://neo4j.com/, 2014. Viitattu 3.1.2015.

[Noiumkar and Chomsiri, 2014] Noiumkar, P. and Chomsiri, T., A Comparison the Level of Security on Top 5 Open Source NoSQL Databases. *The 9th International Conference on Information Technology and Applications (ICITA2014)*, 1-4 July, 2014, Sydney, Australia

[NuoDB, 2015] NuoDB-tietokanta. http://www.nuodb.com/, 2015. Viitattu 14.5.2015.

[Okman et al., 2011] Okman, L., Gal-Oz, N., Gonen, Y. and Gudes, E., Security Issues in NoSQL Databases. *International Joint Conference of IEEE TrustCom-11/IEEE ICESS-11/FCST-11,* 2011

[Oracle, 2015] Oracle Database 12c. https://www.oracle.com/database/index.html, 2015. Viitattu 15.5.2015.

[OrientDB, 2015] OrientDB-tietokanta. http://orientdb.com/orientdb/, 2015. Viitattu 15.5.2015.

[Parstream, 2014] ParStream version 3.1.13 Manual, Marraskuu 15, 2014.

[Parstream-verkkosivu, 2015] Parstream, Parstream Geo-Distributed Analytics (GDA). https://www.parstream.com/product/parstream-geo-distributed-analytics/, 2015. Viitattu 29.4.2015.

[PostgreSQL, 2015] PostgreSQL. http://www.postgresql.org/, 2015. Viitattu 15.5.2015

[Pritchett, 2008] Pritchett, D., Base: An acid alternative. Queue **6**(3) (2008): 48-55.

[RavenDB, 2015] RavenDB-tietokanta. http://ravendb.net/, 2015. Viitattu 15.5.2015.

[Redis, 2015] Redis-tietokanta. http://redis.io/, 2015. Viitattu 3.1.2015.

[Reese, 2000] Reese, G., *Database Programming with JDBC and Java, Second edition*. O'Reilly & Associates, Inc., 101 Morris Street, Sebastopol, 2000.

[Riak, 2015] Riak-tietokanta. http://basho.com/riak/, 2015. Viitattu 13.5.2015.

[Rockoff, 2011] Rockoff, L., The Language of SQL. Course Technology, a part of Cengage Learning, 2011.

[ScaleArc, 2015] ScaleArc-tietokantaratkaisu. http://scalearc.com/, 2015. Viitattu 15.5.2015.

[ScaleBase, 2015] ScaleBase-tietokantaratkaisu. https://www.scalebase.com/, 2015. Viitattu 15.5.2015.

[SimpleDB, 2015] Amazon SimpleDB. http://aws.amazon.com/simpledb/, 2015. Viitattu 14.5.2015.

[SQL Server Express, 2015] Microsoft SQL Server Express. http://www.microsoft.com/en-us/server-cloud/products/sql-server-editions/sql-server-express.aspx, 2015. Viitattu 15.5.2015.

[Sqrrl, 2015] Sqrrl-tietokanta. http://sqrrl.com/, 2015. Viitattu 15.5.2015.

[Stonebraker and Çetintemel, 2005] Stonebraker M. and Cetintemel U., "One Size Fits All": An Idea Whose Time Has Come and Gone. *Proceedings of the 21st International Conference on Data Engineering*, p.2-11, April 05-08, 2005

[Strozzi, 2010], Strozzi, C., NoSQL - A Relational Database Management System, http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql/Home%20Page, 2010. Viitattu 29.4.2015.

[TokuDB, 2015] TokuDB: High-Performance Storage Engine for MySQL, MariaDB and Percona Server. http://www.tokutek.com/tokudb-for-mysql/, 2015. Viitattu 15.5.2015.

[Ullman, 1988] Ullman, J., D., *Principles of Database and Knowledge-Base Systems, Vol. I.* Computer Science Press, Inc., New York, NY, USA, 1988.

[Venkatesh, 2012] Venkatesh, P., NewSQL - The New Way to Handle Big Data. http://www.opensourceforu.com/2012/01/newsql-handle-big-data/, 2012. Viitattu 29.4.2015.

[Voldemort, 2015] Project Voldemort: A distributed database. http://www.project-voldemort.com/voldemort/, 2015. Viitattu 14.5.2015.

[VoltDB, 2015] VoltDB-tietokanta. http://voltdb.com/, 2015. Viitattu 14.5.2015.

[W3Computing, 2015] A Comparison of Oracle and MySQL Server. http://www.w3computing.com/sqlserver/comparison-oracle-db2-mysql-sql-server/, 2015. Viitattu 18.2.2015.

[Wiggins, 2010] Wiggins, A., Graph Databases. http://adam.herokuapp.com/past/2010/3/15/graph_databases, 2010. Viitattu 27.5.2015.

[Wong, 2014] Wong, M., Current Data Security Issues of NoSQL Databases. Network Defense & Forensics Insights, Fidelis Cybersecurity Solutions Inc, 1601 Trapelo Road, Suite 270, Waltham, MA 02451, 2014

[XAP, 2015] XAP - In-Memory Computing Platform. http://www.gigaspaces.com/xap-in-memory-computing-event-processing/Meet-XAP, 2015. Viitattu 15.5.2015.

[Zahid et al, 2014] Zahid, A., Masood, R. and Shibli, A. Security of Sharded NoSQL Databases: A Comparative Analysis. *Conference on Information Assurance and Cyber Security (CIACS)* A performance comparison for Architects and IT Managers. White paper, 2014.